Biblioteczka

Komputer Świat



DVD I KŚ+
- NAJLEPSZE NARZĘDZIA DO PROGRAMOWANIA
- PLIKI SZKOLENIOWE

ISO płyty do pobrania z ksplus.pl

JAVASCRIPT
HTML **CSS** PHP WORDPRESS

TWORZENIE STRON I APLIKACJI WEBOWYCH
KURS PROGRAMOWANIA OD PODSTAW



KROK PO KROKU OPANUJESZ:

- najprostsze sposoby tworzenia stron WWW – własny blog
- ABC WordPressa
- podstawy programowania w JavaScripcie
- tworzenie własnych aplikacji webowych



Z TĄ KSIĄŻKĄ E-WYDANIE GRATIS

Poniżej znajduje się płyta z kodem bonusowym dającym dostęp do e-wydania tej książki w serwisie KŚ+ (ksplus.pl) oraz pliku ISO z cyfrową wersją płyty do pobrania.

NA PŁYCIE DVD

Płyta dołączona do tej książki zawiera narzędzia przydatne do programowania stron internetowych i aplikacji webowych: edytory kodu źródłowego, serwery, programy do FTP, edytory WWW, WordPressa.

Jeżeli brakuje płyty, poinformuj sprzedawcę lub redakcję: pomoc@komputerswiat.pl

JAVA SCRIPT

PHP

HTML

CSS

PŁYTA JEST DODATKIEM DO KSIĄŻKI

KOMPUTER ŚWIAT BIBLIOTECZKA

4|2022

Komputer Świat

NA PŁYCIE ZNAJDZIESZ KOD DO E-WYDANIA I CYFROWEJ WERSJI PŁYTY W KSPLUS.PL

Kod bonusowy należy zarejestrować w KŚ+ (ksplus.pl)

Biblioteczka Komputer Świat

KRZYSZTOF DZIEDZIC

JAVASCRIPT
HTML **CSS** PHP
WORDPRESS

TWORZENIE STRON I APLIKACJI WEBOWYCH
KURS PROGRAMOWANIA OD PODSTAW

ringier
axel springer

AUTOR: Krzysztof Dziedzic

REDAKTORZY PROWADZĄCY: Rafał Kamiński, Agnieszka Al-Jawahiri

PRZYGOTOWANIE PŁYTY: Mariusz Michalski

PROJEKT OKŁADKI: Robert Dobrzyński

SKŁAD I ŁAMANIE: Mariusz Rybak

KOREKTA: Jolanta Rososińska

WYDAWCA:

RINGIER AXEL SPRINGER POLSKA Sp. z o.o.
02-672 Warszawa, ul. Domaniewska 49
tel. 12 2600200 (BOK)
www.ringieraxelspringer.pl

ISBN 978-83-8250-195-7

Warszawa 2022

BUSINESS PROJECT MANAGER: Paweł Bulwan

DRUK I OPRAWA:
Drukarnia im. Adama Półtawskiego, Kielce

EGZEMPLARZE ARCHIWALNE:
literia.pl, prenumerata.axel@qg.com

E-WYDANIA, E-PRENUMERATA:
ksplus.pl

KONTAKT:
redakcja@komputerswiat.pl

INTERNET:
komputerswiat.pl, ksplus.pl

Płyta DVD jest dodatkiem do książki

ringier
axel springer

1 Co warto wiedzieć na początek

Zanim w kolejnych rozdziałach zajmiemy się tworzeniem stron internetowych i aplikacji webowych z wykorzystaniem języka JavaScript, poznajmy inne bardzo ważne technologie, bez których nie dałoby się w praktyce stosować JavaScriptu. Są to HTML, CSS i PHP

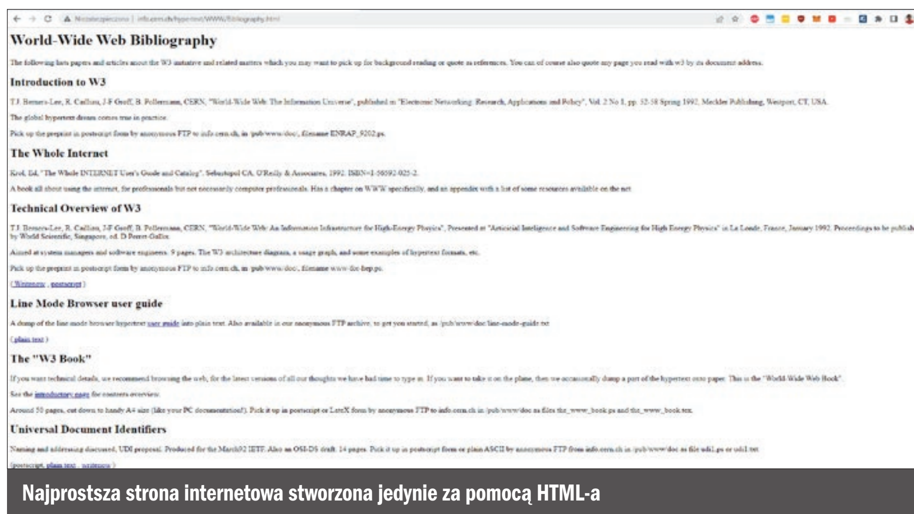
Bardzo często na stronach tej książki będziemy używać nazwy JavaScript w odniesieniu do języka programowania na potrzeby aplikacji i stron internetowych. Popularnie określa się go po prostu skrótem od JavaScript – JS. Będziemy tych dwóch wyrażeń używać zamiennie. Celem tej książki jest zapoznanie Czytelników z podstawami programowania w JavaScripcie oraz innych technologiach internetowych, aby w przyszłości mogli realizować projekty webowe na własne potrzeby.

Strona internetowa – czym jest?

W dużym skrócie jest to zbiór różnego typu elementów, które powinny być logicznie uporządkowane w celu wyświetlania w prze-

POZNAJMY HTML

HTML (HyperText Markup Language) to język znaczników, który służy do tworzenia dokumentów hipertekstowych. Pozwala opisać strukturę informacji zawartych wewnątrz strony internetowej. Dzięki znacznikom możemy nadać odpowiednie znaczenie konkretnym fragmentom tekstu i w ten sposób formować hiperłącza, akapity, nagłówki, listy itp. HTML umożliwia określenie wyglądu dokumentu w przeglądarce internetowej. Do szczegółowego opisu formatowania akapitów, nagłówków, czcionek, kolorów zalecane jest jednak korzystanie z CSS.



World-Wide Web Bibliography

The following lists papers and articles about the W3 initiative and related matters which you may want to pick up for background reading or quote as references. You can of course also quote any page you read with w3 by its document address.

Introduction to W3

T.J. Berners-Lee, R. Cailliau, J-F Groff, B. Pollermann, CERN, "World-Wide Web: The Information Universe", published in "Electronic Networking: Research, Applications and Policy", Vol. 2 No 1, pp. 52-58 Spring 1992, Meckler Publishing, Westport, CT, USA.

The global hypertext dream comes true in practice.

Pick up the preprint in postscript form by anonymous FTP to info.cern.ch, in /pub/www/doc/, filename ENRAP_9202.ps.

The Whole Internet

Krol, Ed, "The Whole INTERNET User's Guide and Catalog". Sebastopol CA. O'Reilly & Associates, 1992. ISBN=1-56592-025-2.

A book all about using the internet, for professionals but not necessarily computer professionals. Has a chapter on WWW specifically, and an appendix with a list of some resources available on the net.

Technical Overview of W3

T.J. Berners-Lee, R. Cailliau, J-F Groff, B. Pollermann, CERN, "World-Wide Web: An Information Infrastructure for High-Energy Physics", Presented at "Artificial Intelligence and Software Engineering for High Energy Physics" in La Londe, France, January 1992. Proceedings to be published by World Scientific, Singapore, ed. D Perret-Gallix.

Aimed at system managers and software engineers. 9 pages. The W3 architecture diagram, a usage graph, and some examples of hypertext formats, etc.

Pick up the preprint in postscript form by anonymous FTP to info.cern.ch, in /pub/www/doc/, filename www-for-hep.ps.

(Writeup , postscript)

Line Mode Browser user guide

A dump of the line mode browser hypertext user guide into plain text. Also available in our anonymous FTP archive, to get you started, as /pub/www/doc/line-mode-guide.txt

(plain text)

The "W3 Book"

If you want technical details, we recommend browsing the web, for the latest versions of all our thoughts we have had time to type in. If you want to take it on the plane, then we occasionally dump a part of the hypertext onto paper. This is the "World Wide Web Book".

See the introductory page for contents overview.

Around 50 pages, cut down to handy A4 size (like your PC documentation!). Pick it up in postscript or LaTeX form by anonymous FTP to info.cern.ch in /pub/www/doc as files the_www_book.ps and the_www_book.tex.

Universal Document Identifiers

Naming and addressing discussed, UDI proposal. Produced for the March92 IETF. Also an OSI-DS draft. 14 pages. Pick it up in postscript form or plain ASCII by anonymous FTP from info.cern.ch in /pub/www/doc as file udi1.ps or udi1.txt

(postscript, plain text , writeup)

Najprostsza strona internetowa stworzona jedynie za pomocą HTML-a

glądarce użytkownikowi końcowemu. Elementy na stronie internetowej połączone są za pomocą hiperlinków. Strony internetowe mogą być zakładane w celu spełniania różnorodnych zadań, na przykład prowadzenia bloga, udostępnienia informacji na temat naszej firmy itp. Głównym ich celem jest prezentacja pewnego rodzaju danych. Charakter stron jest właściwie zawsze pasywny i można powiedzieć, że głównie zawiera jedynie część wizualną – frontową, inaczej nazywaną frontend.

Aplikacja webowa – czym jest?

Wbrew temu, co często się sądzi, aplikacja webowa nie jest zamiennikiem strony internetowej, ale jej alternatywą. Jest to element pośredni pomiędzy stroną internetową a aplikacją, którą uruchamiamy w konkretnym systemie operacyjnym. Tak więc precyzyjnie

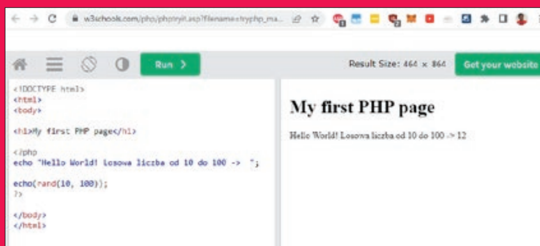


Strona ksplus.pl po zalogowaniu na konto użytkownika – jest to przykład interaktywnej aplikacji webowej

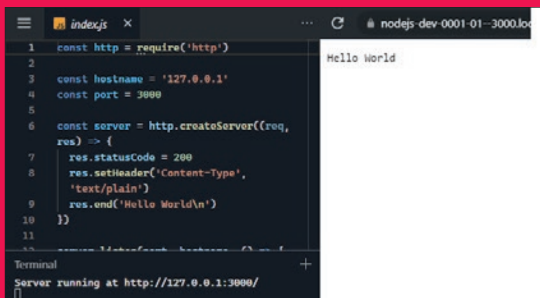
TECHNOLOGIE, KTÓRE SĄ WYKORZYSTYWANE PRZY TWORZENIU

Strony internetowe, których zadaniem jest jedynie prezentowanie treści, tworzone są za pomocą HTML-a, CSS i JS. Aplikacje webowe natomiast wymagają również warstwy backendowej, która jest połączona z warstwą frontendową. Użytkownik, który wykonuje jakąś akcję na części frontowej, powoduje przesłanie żądania do części backendowej i otrzymuje z niej odpowiedź wywołującą wyświetlenie konkretnej informacji. Poniżej zostały opisane w kilku zdaniach różnego typu technologie internetowe, które pomagają przy tworzeniu aplikacji webowych.

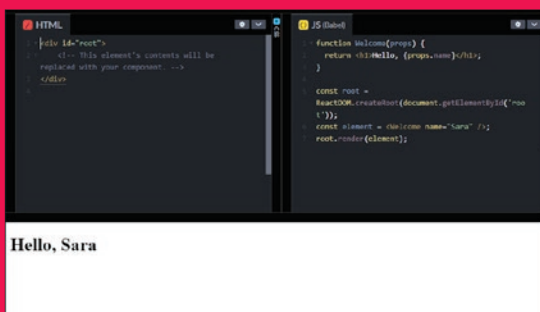
■ **PHP** – jest to jedna z bardziej popularnych, choć można również powiedzieć – wysłużonych technologii do tworzenia dynamicznych aplikacji internetowych. Zaletą PHP jest łatwość wdrażania i ogromna ilość gotowych bibliotek i frameworków. Aplikacje webowe, które są zbudowane z użyciem PHP, to między innymi Wikipedia i Facebook. Ten ostatni stworzył nową wersję PHP, która pozwala zarządzać aplikacją dla ponad 2 miliardów użytkowników ze zdecydowanie lepszą wydajnością niż standardowy PHP. Ta nowa odmiana PHP to Hack na potrzeby HHVM.



■ **Node.js** – to środowisko, które umożliwia uruchamianie kodu JavaScript po stronie serwera. Do zdecydowanych zalet tego środowiska należy asynchroniczność, czyli nieblokująca obsługa wejścia/wyjścia, wysoka skalowalność oraz duża popularność. Korzystają z niego takie firmy, jak Netflix i Uber.

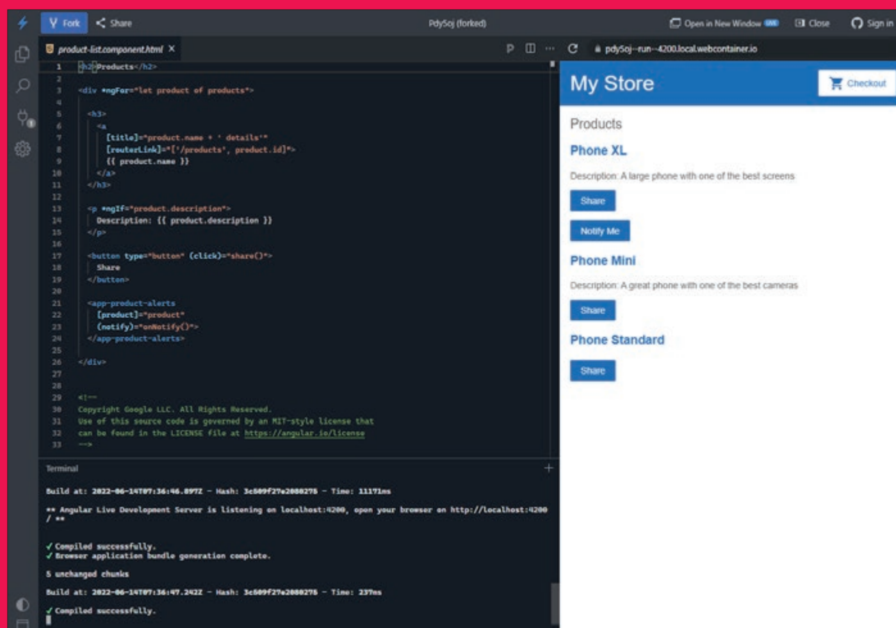


■ **React** – jest to właściwie biblioteka samego JavaScriptu, która umożliwia wygodne tworzenie interfejsu użytkownika, a tym samym ułatwia budowanie dynamicznych i interaktywnych aplikacji webowych. Jako zalety Reacta można wskazać jego gotowe, wysoce konfigurowalne komponenty, które pozwalają na zaoszczędzenie czasu programisty.

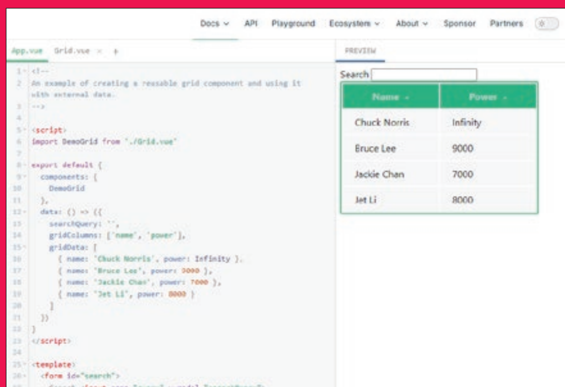


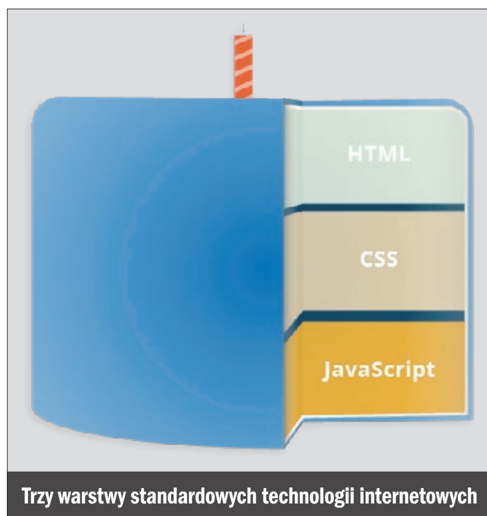
STRON INTERNETOWYCH I APLIKACJI WEBOWYCH

■ **Angular** – ten framework został stworzony przez Google i jest powszechnie używany do tworzenia aplikacji webowych. Głównie stosuje się go do SPA (Single Page Application), czyli aplikacji, które są ładowane w przeglądarce dynamicznie, bez konieczności pobierania dodatkowych danych przy przechodzeniu między poszczególnymi stronami. Angular dostarcza w pakiecie wszystko, co niezbędne do tworzenia aplikacji. Został napisany z użyciem języka TypeScript, w trakcie kompilacji ostatecznie zostaje kompilowany do JavaScriptu. Jest darmowy i każdy może z niego korzystać.



■ **Vue** – jest to framework do tworzenia interfejsów użytkownika. Zdaniem twórcy ma łączyć w sobie najlepsze cechy Reacta oraz Angulara. Vue wykorzystuje virtual DOM, dzięki czemu zyskujemy bardzo dobrą wydajność. Wyróżnia się również małą wagą plików oraz dużą elastycznością. Ma prosty przejrzysty kod oraz dobrze uzupełnioną dokumentację.





Trzy warstwy standardowych technologii internetowych

mówiąc, jest to program dostępny w oknie przeglądarki internetowej. Oznacza to, że aplikacja webowa ma charakter interaktywny – można wykonywać na niej różnego rodzaju akcje.
Aplikacje webowe składają się z kilku warstw. Jedną z nich jest frontend, czyli część wizualna, najczęściej tworzona za pomocą na przykład HTML-a, CSS i JavaScriptu. Aplikacja zawiera też warstwę backendową (kod aplikacji, program wykonywalny), dość często opisany w PHP. Tak naprawdę jednak obecnie kod backendowy aplikacji może być tworzony w praktycznie dowolnym języku, bo dzięki odpowiednim frameworkom, które zapewniają niezbędną integrację i udostępniają potrzebne biblioteki, będzie działać w dowolnej przeglądarce.
Największą zaletą takiego rozwiązania jest to, że aplikacja interaktywna może być dowolnie rozbudowywana i dostępna w dowolnej przeglądarce na dowolnym urządzeniu. Z punktu widzenia biznesowego jest to idealne rozwiązanie, gdyż bez żadnego problemu, wykorzystując jedynie jedną aplikację webową, możemy dotrzeć do ogromnej grupy użytkowników, którzy muszą mieć tylko dostęp do internetu i przeglądarkę.

UWAGA! WYMAGANIA WSTĘPNE

W ramach tej książki skupiamy się głównie na poznaniu i nauce JavaScriptu. W celu pełnego zrozumienia wszystkich wskazówek i instrukcji, które są tu przedstawione, przydatna jest wprawdzie podstawowa wiedza z zakresu HTML-a oraz CSS, ale wszystkie pliki źródłowe zostały zamieszczone na płycie i w KŚ+ (ksplus.pl), tak aby każdy mógł uruchomić u siebie, w środowisku lokalnym, prezentowane tu skrypty nawet bez posiadania podstawowej wiedzy.

POZNAJMY CSS

CSS (Cascading Style Sheets), czyli kaskadowe arkusze stylów, jest to język, który służy do opisu formy prezentacji stron WWW. Arkusz stylów CSS to w skrócie lista reguł, które ustalają, w jaki sposób ma zostać wyświetlona przez przeglądarkę zawartość strony internetowej. CSS został stworzony w celu oddzielenia struktury dokumentu od formy jego prezentacji. Takie rozdzielenie pozwala uzyskać bardziej przejrzysty kod strony, ułatwia wprowadzanie zmian i zwiększa dostępność witryny.

JavaScript – czym jest i do czego jest wykorzystywany?

JavaScript to język programowania, który umożliwia wdrożenie na stronie internetowej skomplikowanych elementów, dzięki którym strona ta może nie tylko wyświetlać statyczne informacje, ale również obsługiwać zmianę treści odpowiednio do sytuacji, wyświetlać interaktywne mapy i animacje grafiki 2D/3D, wideo itd.
Jest to trzecia warstwa standardowych technologii internetowych, poza HTML-em i CSS. Pozwala na tworzenie dynamicznych stron oraz aplikacji webowych.

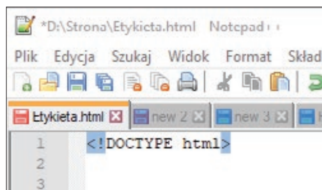
Jak to działa w praktyce? Tworzymy dynamiczną etykietę na stronie

Poznane przez nas trzy podstawowe warstwy układają się jedna na drugiej. Jako prosty przykład posłuży nam dynamiczna etykieta tekstowa. Na początku nie musimy korzystać z rozbudowanych narzędzi deweloperskich do tworzenia witryn ani pisać skomplikowanego kodu. Wystarczy nam Notatnik lub polecany do kodowania **Notepad++** (DVD-KOD: 014/015), który czytelniej prezentuje składnię kodu.
Tak podstawowe narzędzie będzie zupełnie wystarczające, gdyż kod strony internetowej zapisany jest w pliku tekstowym lub, precyzyjniej określając, w pliku z hipertekstem i bez problemu będziemy mogli go tworzyć w dowolnym edytorze tekstowym.

Warstwa HTML

Rozpoczynamy od wprowadzenia tekstu dla tej właśnie warstwy.

1 Pisanie naszej strony z etykietą zaczniemy od linii tekstu – **<!DOCTYPE html>**. W ten sposób informujemy przeglądarkę internetową, że ma do czynienia ze stroną napisaną w HTML-u.

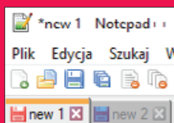


2 Teksty zapisane w nawiasach ostrokątnych nazywamy **znacznikami** (**tagami**). Wiele znaczników ma część otwierającą i zamykającą. Tak jest na przykład ze znacznikiem **html**. Służy on do oznaczenia całości strony. Pokazuje, gdzie się ona zaczyna i gdzie się kończy. Otwierająca część znacznika to jedynie jego nazwa w nawiasie ostrokątnym. Zamykająca dodatkowo przed nazwą zawiera ukośnik. W kolejnych dwóch liniach zapiszmy zatem **<html>** i **</html>**. Oprócz tego od razu również w tagu otwierającym dodajemy deklarację języka witryny – **lang="pl-PL"** – dzięki temu robot wyszuki-

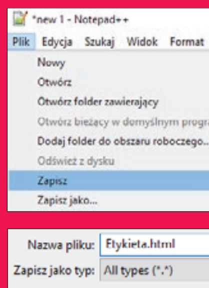
NAJPIERW TWORZYMY ODPOWIEDNI DOKUMENT

Cała napisana podczas realizacji tego przykładu strona będzie dokumentem HTML. Przed rozpoczęciem pisania warto utworzyć w edytorze Notepad++ plik w odpowiednim formacie, by program mógł prawidłowo kolorować tekst i stosować autouzupełnianie zgodne z zasadami używanego języka.

1 Uruchamiamy program Notepad++. Klikamy w górnym lewym rogu na symbol tworzenia nowego dokumentu lub klikamy na **Plik**, **Nowy**.

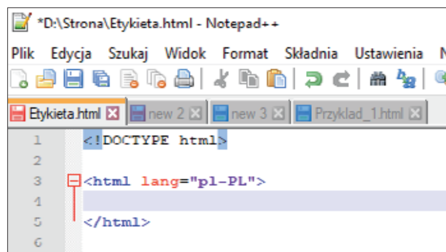


2 Zanim zaczniemy wprowadzać kod, klikamy na **Plik**, **Zapisz** i podajemy nazwę dla nowego pliku, w naszym przykładzie będzie to **Etykieta.html**.

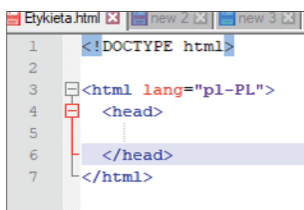


3 Teraz wprowadzane w tym dokumencie wyrażenia i składnia będą interpretowane jako napisane w HTML-u i odpowiednio kolorowane. Dzięki temu kod strony będzie czytelniejszy.

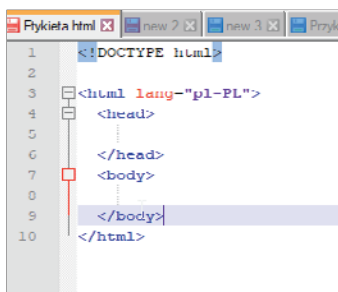
warki będzie w stanie poprawnie ją sklasyfikować. Podajemy tutaj deklarację zgodną ze standardem HTML5.



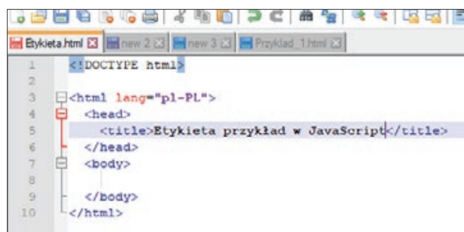
3 Dalsze znaczniki będziemy zapisywać wewnątrz znacznika html. To znaczy – po znaczniku otwierającym, ale przed znacznikiem zamykającym. W ten właśnie sposób konstruuje się strukturę dokumentu. Wnętrze takiego znacznika **html** powinno składać się z dwóch sekcji – **head** i **body**,



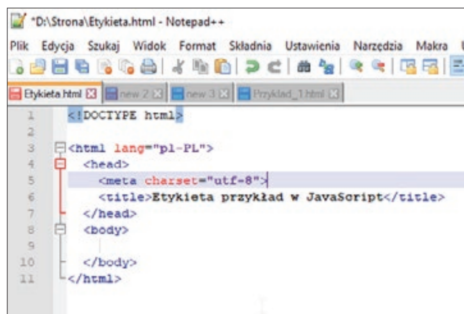
czyli nagłówka i treści. Najpierw otwieramy i zamykamy znacznik head, a dalej powinno znaleźć się otwarcie i zamknięcie znacznika body. Head to nagłówek. W tej części zapisujemy treści bardziej techniczne, dotyczące działania dokumentu. A w części body skupiamy się na treści, czyli na tym, co widać na stronie.



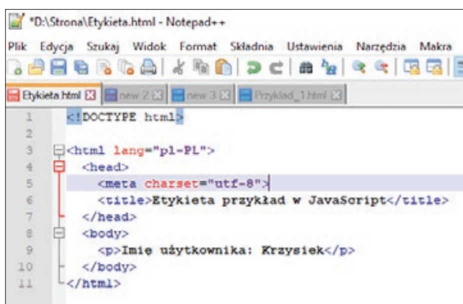
4 W sekcji head można między innymi określić, jaki ma być napis na pasku tytułowym strony. Służy do tego znacznik **title**. Otwieramy więc znacznik, wpisujemy tytuł, a następnie zamykamy znacznik.



5 Istotnym elementem, jakiego można użyć w sekcji **head**, jest znacznik **meta**, który pozwala na określenie metadanych. Jedną z możliwych i najczęściej stosowanych form użycia tego znacznika jest **<meta charset ="utf-8">** – przy tym znaczniku nie musimy stosować zamknięcia. Ten zapis pozwala na ustawienie kodowania strony z wykorzystaniem Unicode. Bez użycia tego znacznika niektóre z przeglądarek internetowych mogą mieć problem z prawidłowym wyświetlaniem polskich znaków na stronie.

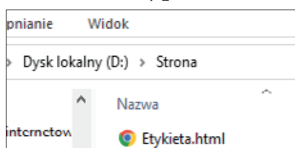


6 W sekcji **body** umieszczamy faktyczną treść strony. Możemy zapisać ją wewnątrz znacznika **p**, co oznacza paragraf (akapit), lub nagłówków tekstowych **h1, h2** itp. Treści umieszczone w sekcji **body** są wyświetlane na stronie w takiej kolejności, w jakiej są tam zapisane. Jeśli chcemy wstawiać na swoją stronę kolejne wpisy, by były widoczne u góry strony,

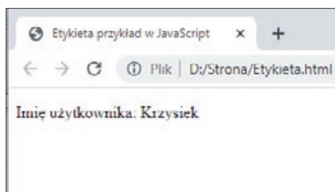


jako treści aktualne, można dopisywać je na samym początku sekcji body.

7 Teraz zapisujemy nasz dokument, klikając na **Zapisz** lub korzystając ze skrótu ctrl+S. Przechodzimy do folderu, w którym znajduje się nasz plik, i klikamy na niego dwukrotnie – powinna zostać uruchomiona nasza domyślna przeglądarka z lokalnym podglądem wskazanej przez nas strony.



8 W widoku strony możemy bez problemu sprawdzić, że nadany przez nas tytuł jest widoczny oraz stworzony przez nas paragraf zawiera podaną przez nas treść. Dodatkowo wszystkie polskie znaki zostały poprawnie wyświetlone dzięki temu, że określiliśmy kodowanie strony.

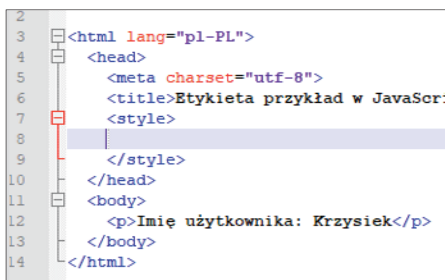


Jest to czysta strona HTML, która jak łatwo zauważyć, jest zupełnie pasywna – możemy jedynie odczytać informacje zapisane w kodzie źródłowym strony. Czas dodać warstwę CSS, która umożliwia zarządzanie wyglądem poszczególnych sekcji.

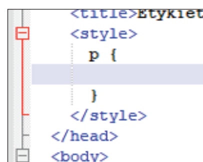
Warstwa CSS

Podstawowa treść strony została już zapisana. Mając już podstawową treść strony, można przejść do określenia wyglądu dla poszczególnych jej elementów. W naszym prostym przykładzie mamy jedynie sekcję **<p>**, czyli paragraf, któremu możemy nadać styl.

1 W celu dodania formatowania CSS do naszej strony należy w sekcji **head** umieścić znacznik **style**.



2 Wewnątrz niego zapisujemy nazwy znaczników, dla których chcemy zmienić wygląd. My chcemy zmienić styl tekstów dla treści zapisanych w znacznikach p. Dlatego nazwę tego znacznika umieszczamy w sekcji style. By dla znacznika zmieniać poszczególne właściwości dotyczące wyglądu, trzeba wpisywać ich nazwy w nawiasie klamrowym.

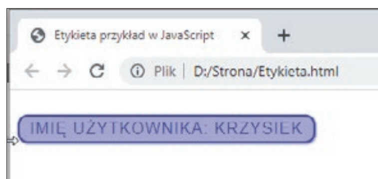


3 Ponieważ jest to jedynie przykład, nie będziemy zgłębiać poszczególnych wyrażeń służących do opisu stylu dla poszczególnych znaczników. W naszym przykładzie użyjemy aż 12 reguł – jeśli chcemy dowiedzieć się, co każdy z nich oznacza, możemy uzyskać więcej informacji na stronie **developer.mozilla.org/pl/docs/Web/CSS**

4 Po dodaniu tekstu tak jak na ilustracji na kolejnej stronie wystarczy zapisać nasz dokument i ponownie otworzyć w domyślnej przeglądarce. Jak widać, dodane przez nas style dla paragrafu zadziałały, mamy już

```
<head>
  <meta charset="utf-8">
  <title>Etykieta przykład w JavaScript</title>
  <style>
    p {
      font-family: 'helvetica neue', helvetica, sans-serif;
      letter-spacing: 1px;
      text-transform: uppercase;
      text-align: center;
      border: 2px solid rgba(0,0,200,0.6);
      background: rgba(0,0,200,0.3);
      color: rgba(0,0,200,0.6);
      box-shadow: 1px 1px 2px rgba(0,0,200,0.4);
      border-radius: 10px;
      padding: 3px 10px;
      display: inline-block;
      cursor: pointer;
    }
  </style>
</head>
<body>
  <p>Imię użytkownika: Krzysiek</p>
```

na naszej stronie dwie warstwy – HTML oraz CSS – które pozwalają na uzyskanie ciekawego designu.
Do tego momentu tworzona przez nas strona nie była interaktywna – czas dodać do niej element dynamiczny, z którym będziemy mogli wchodzić w interakcje. W tym celu musimy dodać kolejną warstwę – JavaScript.



Warstwa JavaScript

Do tej pory przedstawione przez nas warstwy HTML oraz CSS miały na celu jedynie zobrazowanie, jak krok po kroku tworzone są strony internetowe. Teraz przechodzimy do najciekawszej warstwy, która pozwala już na pisanie złożonego kodu logicznego, w ramach którego możemy wchodzić w interakcję z użytkownikiem i tworzyć dynamiczne strony oraz aplikacje webowe. W ramach naszego przykładu **Etykieta.html** to strona internetowa, która wyświetla sformatowany paragraf, który wyglądem przypomina przycisk. Zawiera on informacje o imieniu użytkownika. Dzięki JavaScriptowi dodamy teraz możliwość klikania na ten przycisk i sprawimy, by po wykonaniu tej akcji można było podać nową wartość, która będzie prezentowana na stronie.

1 Rozpoczynamy od dodania znacznika **script** w sekcji **body**. Warto wiedzieć, że znacznik script możemy również umieszczać wewnątrz innych sekcji, na przykład head.

```
    }
  </style>
</head>
<body>
  <p>Imię użytkownika: Krzysiek</p>

  <script>

  </script>
</body>
</html>
```

2 Teraz chcemy, aby użytkownik mógł wchodzić w interakcję z paragrafem dodanym powyżej. W tym celu definiujemy zmienną **paragraf**, która będzie wskazywać na paragraf dodany w kodzie HTML.

```
<body>
  <p>Imie użytkownika: Krzysiek</p>

  <script>
    const paragraf = document.querySelector('p');
```

3 Następnie dodajemy obsługę zdarzenia kliknięcia na paragraf poprzez wykorzystanie metody **addEventListner**. Definiuje-

my, że po kliknięciu ma zostać uruchomiona funkcja **zmienNazwe**.

```
</head>
<body>
  <p>Imię użytkownika: Krzysiek</p>

  <script>
    const paragraf = document.querySelector('p');

    paragraf.addEventListener('click', zmienNazwe);
```

4 Pozostaje nam dodać wspomnianą funkcję. Wprowadzamy kolejną zmienną, która

```
    function zmienNazwe() {
      const name = prompt('Podaj nowe imię');
      paragraf.textContent = `Imię użytkownika: ${name}`;
    }
  </script>
 </body>
</html>
```

będzie odpowiedzialna za wywoływanie okna z prośbą o wprowadzenie tekstu, i procedurę zmiany imienia widocznego w naszym paragrafie na tekst wprowadzony w wyskakującym oknie.

5 Cała strona wygląda następująco od strony kodu:

```
<!DOCTYPE html>

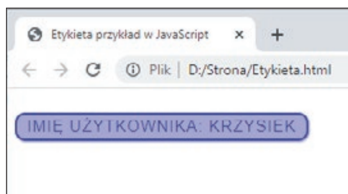
<html lang="pl-PL">
  <head>
    <meta charset="utf-8">
    <title>Etykieta przykład w JavaScript</title>
    <style>
      p {
        font-family: 'helvetica neue', helvetica, sans-serif;
        letter-spacing: 1px;
        text-transform: uppercase;
        text-align: center;
        border: 2px solid rgba(0,0,200,0.6);
        background: rgba(0,0,200,0.3);
        color: rgba(0,0,200,0.6);
        box-shadow: 1px 1px 2px rgba(0,0,200,0.4);
        border-radius: 10px;
        padding: 3px 10px;
        display: inline-block;
        cursor: pointer;
      }
    </style>
  </head>
  <body>
    <p>Imię użytkownika: Krzysiek</p>

    <script>
      const paragraf = document.querySelector('p');

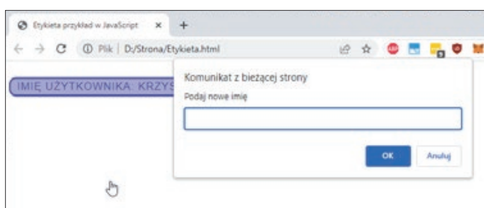
      paragraf.addEventListener('click', zmienNazwe);

      function zmienNazwe() {
        const name = prompt('Podaj nowe imię');
        paragraf.textContent = `Imię użytkownika: ${name}`;
      }
    </script>
  </body>
</html>
```

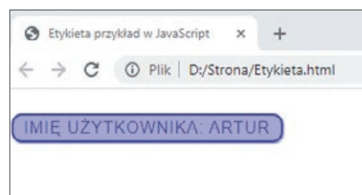
6 W celu sprawdzenia działania stworzonej przez nas funkcjonalności zapisujemy plik i otwieramy go w przeglądarce.



7 Na pierwszy rzut oka wszystko wygląda tak samo, jednak po kliknięciu na nasz przycisk – w tym wypadku paragraf – pojawia się okno z prośbą o podanie tekstu.



8 Po wprowadzeniu nowego tekstu zawartość strony zostanie dynamicznie zmieniona.



Jest to jeden z najprostszych przykładów wykorzystania HTML, CSS oraz JS do stworzenia interaktywnej strony. Oczywiście można tworzyć zdecydowanie bardziej skomplikowane strony. W kolejnym rozdziale zobaczymy, w jaki sposób można szybko tworzyć dynamiczne strony bez zbędnego wysiłku – niejako idąc na skróty i dodatkowo bez znajomości języków programowania.

2 Idziemy na skróty – własny blog lub strona od ręki

Stworzyć stronę WWW czy aplikację webową zupełnie od podstaw to dość trudna sprawa. Przeczytajmy więc na początek, jak ułatwić sobie to zadanie, posłużyć się sposobem i praktycznie bez żadnej wiedzy szybko założyć bloga i własną stronę

Wielu użytkowników rozpoczyna swoją przygodę z tworzeniem stron od bloga lub prostej witryny zarządzalnej z poziomu interfejsu graficznego.
W takim wypadku nie jest konieczna praktycznie żadna wiedza z zakresu technologii internetowych, a informacje umieszczone w pierwszym rozdziale świetnie sprawdzą się jako wprowadzenie.

Tworzenie strony i jej cel

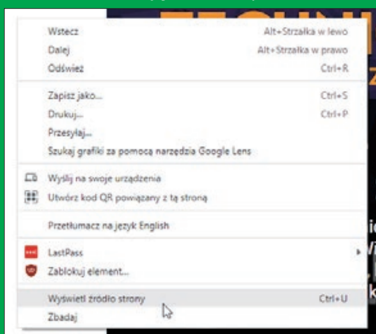
Przy tworzeniu strony WWW ważny jest pomysł. Należy zadać sobie pytanie, co nasza strona ma zawierać, o czym ma być, do jakich osób chcemy dotrzeć i co przez to osiągnąć. Zdecydowanie najlepiej jest tworzyć strony na tematy, które nas interesują. Jeśli spełnimy ten warunek, będzie nam łatwiej, bo dodawanie treści i dbanie o wygląd witryny będzie sprawiało nam przyjemność i pozwoli nam rozwijać pasję i dzielić ją z ludźmi o podobnych zainteresowaniach. Wtedy praca nad stroną będzie bardziej przyjemnością niż pracą.
Chociaż, oczywiście, im bardziej będziemy zagłębiać się w temat tworzenia stron, tym więcej będzie pojawiać się trudności i z czasem okaże się, że potrzebujemy bardziej zaawansowanych narzędzi – poznamy je w kolejnych rozdziałach.

HISTORIA STRON WWW

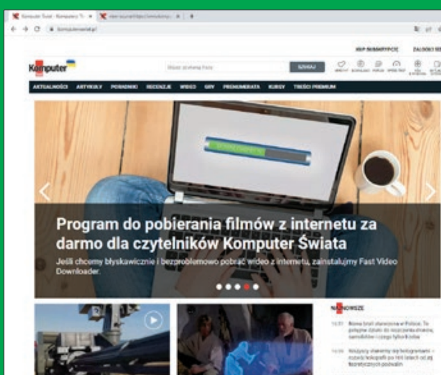
Tworzenie stron internetowych ma za sobą już całkiem długą historię. Właściwie już ponad 30 lat. Pierwszą stronę internetową stworzył około 1990 roku Tim Berners-Lee – jeden z twórców całej usługi WWW, czyli ogólnodostępnych stron internetowych. Jej adres to **info.cern.ch** i nadal można ją odwiedzić. Od tego czasu na świecie pojawiły się ponad 2 miliardy stron – aktywnych na co dzień pozostaje ponad 100 tysięcy. Ma to związek z koniecznością płacenia za domeny i utrzymywania serwerów. Jeśli ktoś znudzi się swoją witryną i nie opłaci domeny – jego strona przestaje istnieć w sieci.
Nie będziemy jednak zagłębiali się w to, w jaki sposób strony były tworzone kiedyś – ważne jest to, jak tworzone są obecnie.

SPRAWDZAMY KOD STRONY

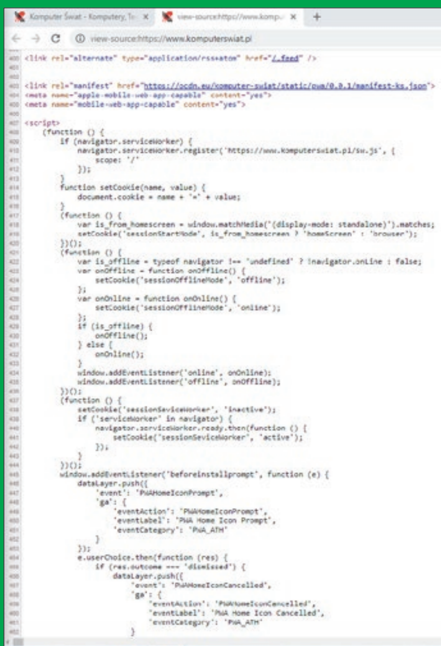
Wszyscy korzystamy z przeglądarek graficznych, więc strona internetowa jest dla nas po prostu graficzną interpretacją kodu, który jest wszyty w backendzie strony. To właśnie interpretując ten kod, przeglądarka zamienia setki linijek tekstu z różnymi znaczkami na czytelną dla nas witrynę, która jest interaktywna.
Jeśli jesteśmy ciekawi, jak wygląda kod strony internetowej, którą aktualnie przeglądamy, możemy to łatwo sprawdzić. Oto, jak to zrobić za pomocą przeglądarki Google Chrome (w innych przeglądarkach opisane kroki są podobne).



1 Uruchamiamy przeglądarkę i wchodzimy na witrynę, której kod chcemy sprawdzić.



2 Następnie klikamy prawym przyciskiem myszy na wolną przestrzeń strony i wybieramy z menu dialogowego opcję Wybierz źródło strony.



3 Strona główna witryny **komputerswiat.pl** ma ponad 6000 linii kodu. Jest to bardzo rozbudowana witryna, która ma w sobie osadzone różnego rodzaju odnośniki, grafiki, filmy oraz reklamy. Nasza pierwsza strona nie będzie aż tak efektowna, ale przeglądanie różnych witryn może być inspirujące.

Proste narzędzie na początek

Nowe strony są tworzone za pomocą HTML5, CSS 3, JavaScriptu. Ale jeśli nie znamy się w ogóle na programowaniu, nie musimy załamywać rąk.

Początkujący mają do wyboru całkiem sporo łatwych w obsłudze narzędzi.
Najprościej jest skorzystać z serwisu blogowego, takiego jak **Blogger**, opisanego na kolejnych stronach. To kreator typu SaaS, czyli

oprogramowanie jako usługa działająca online, w oknie przeglądarki. W przypadku takich kreatorów użytkownik nie musi niczego instalować po swojej stronie i wykonuje wszelkie operacje przez internet, a serwer, pliki i inne dane znajdują się w chmurze usługodawcy. Bardziej zaawansowanym narzędziem, wciąż jednak ogromnie ułatwiającym tworzenie stron internetowych, jest **WordPress** omówiony w kolejnym rozdziale. WordPress jest najpopularniejszym i darmowym CMS-em, czyli systemem zarządzania treścią. Gdy korzystamy z takiego narzędzia, w ogóle nie interesuje nas kod strony, musimy jedynie wprowadzać tekst i nim zarządzać, ewentualnie dodając elementy graficzne. Takie rozwiązanie pozwala na utworzenie własnej strony szybko, sprawnie i bez znajomości programowania.

Zdecydowanie najbardziej popularne w wypadku blogów są narzędzia typu CMS, choć podejście SaaS zdobywa coraz więcej zwolenników ze względu na brak konieczności utrzymywania serwera.

Pisz o swoich pasjach – na swój sposób

Stwórz niepowtarzalnego, pięknego bloga. To proste i nic nie kosztuje.

UTWÓRZ BLOGA

Mój blog o gotowaniu

Czwartek, 16 stycznia

Przepis na pyszny deser

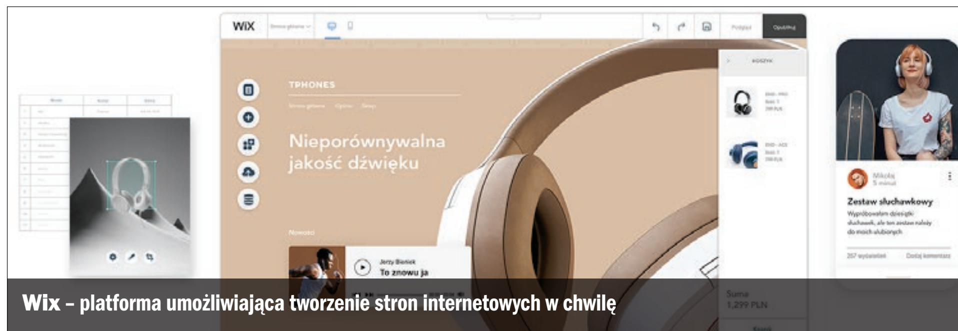
Blogger – narzędzie do tworzenia blogów, pozwala na proste zarządzanie treścią i umożliwia analizę ruchu

Piękne projekty graficzne, potęga dostępnych funkcji i wolność nieskrępowanej kreacji. WordPress jest równie bezcenny co bezpłatny.

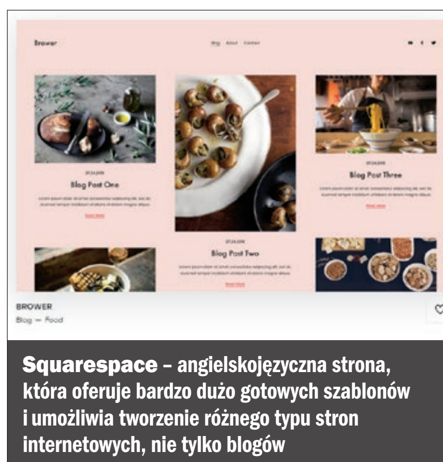
Themes

The works of Berthe Morisot, 1800s-era French painter

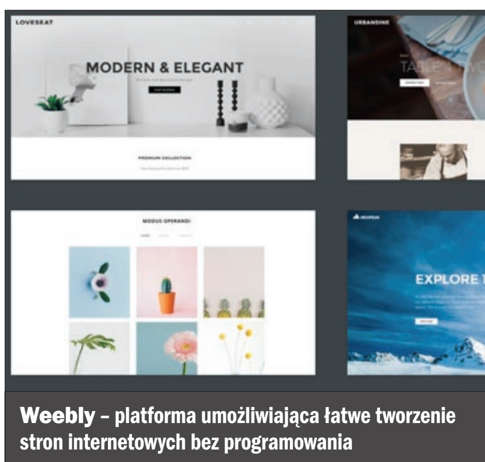
WordPress – jedno z najpopularniejszych narzędzi typu CMS do tworzenia stron internetowych



Wix – platforma umożliwiająca tworzenie stron internetowych w chwilę



Squarespace – angielskojęzyczna strona, która oferuje bardzo dużo gotowych szablonów i umożliwia tworzenie różnego typu stron internetowych, nie tylko blogów



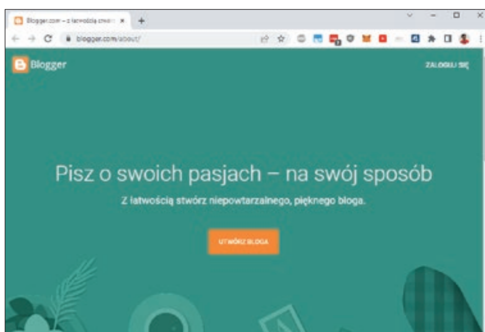
Weebly – platforma umożliwiająca łatwe tworzenie stron internetowych bez programowania

Tworzymy bloga krok po kroku

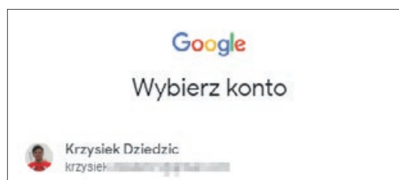
Zakładamy, że chcemy stworzyć bloga o tematyce kulinarnej, który ma być prosty w tworzeniu i utrzymaniu. Takie zadanie łatwo zrealizujemy za pomocą serwisu **Blogspot**.

Uwaga! Stronę w serwisie **Blogger** najlepiej tworzyć w przeglądarce Chrome, Firefox, Safari lub MS Edge. Są to przeglądarki oficjalnie wspierane i zatwierdzone przez twórców.

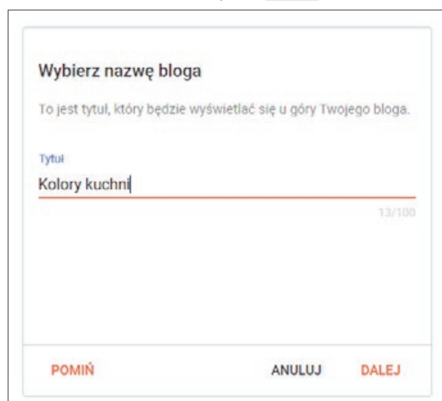
1 Otwieramy jedną ze wspieranych przeglądarek, na przykład Google Chrome, i wchodzimy na stronę: **https://www.blogger.com/about**. Po załadowaniu strony klikamy na **Utwórz bloga** na środku ekranu.



2 Następnie logujemy się za pomocą konta Google. Jeśli go nie mamy, musimy najpierw je założyć.



3 W kolejnym oknie podajemy tytuł naszego bloga i klikamy na **Dalej**.

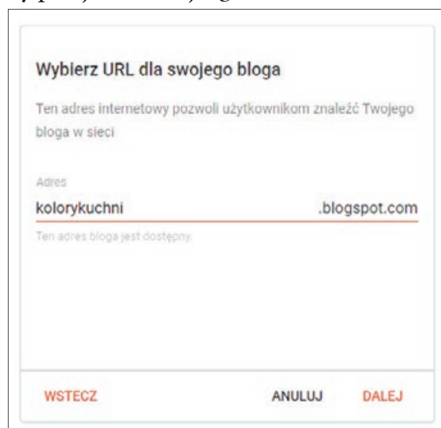


4 Następnie musimy podać adres naszej strony – jest to bardzo ważny krok, gdyż to właśnie po tym adresie inni użytkownicy sieci będą mogli trafić na naszego bloga. Dobrym pomysłem jest zawarcie w adresie tytułu bloga. Jeżeli podany adres będzie dostępny, możemy kliknąć na przycisk **Dalej**, by przejść do kolejnego kroku.

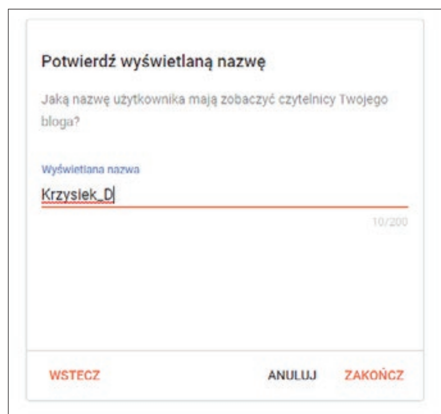
ZARZĄDZANIE BLOGIEM

Od teraz po wejściu na stronę **blogger.com** zostaniemy przeniesieni do panelu zarządzania blogami. Na tym ekranie kliknięcie po lewej stronie na nazwę naszego bloga spowoduje rozwinięcie listy naszych blogów. Możemy dodać kolejny blog, klikając na **Nowy blog**.



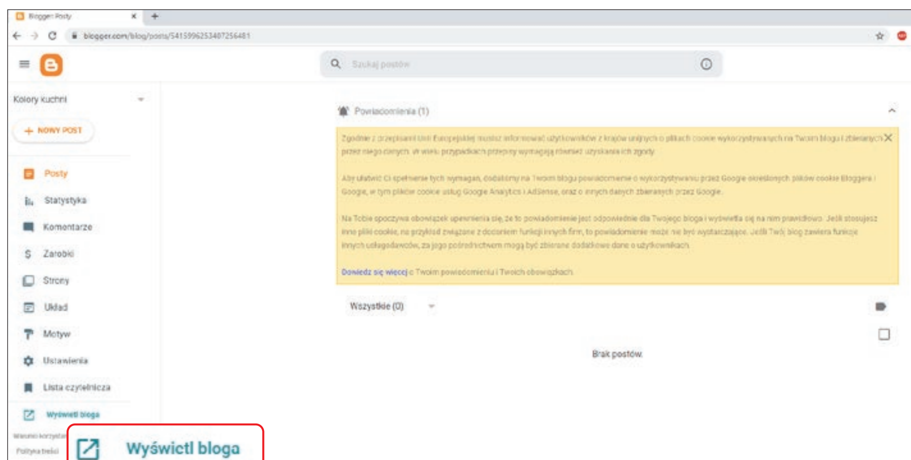


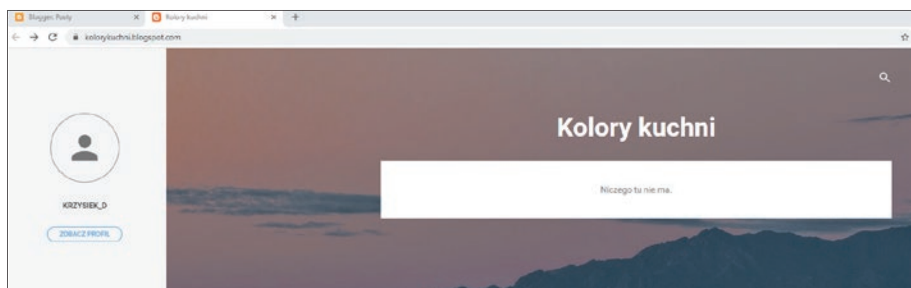
5 Określamy naszą nazwę użytkownika – to pod tą nazwą nasze wpisy poznają czytelnicy bloga. Po podaniu nazwy klikamy na **Zakończ**.



6 Po chwili nasz blog zostanie utworzony i zostaniemy przeniesieni do panelu zarządzania treścią.

7 Po kliknięciu na polecenie **Wyświetl bloga** po prawej stronie ekranu zostanie wyświetlona nasza strona. Domyślnie nasz



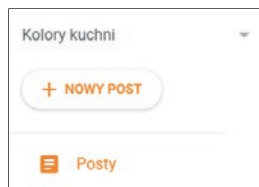


blog będzie zawierał jedynie tytuł oraz informację o wykorzystywaniu plików cookie. Na tym etapie już będzie on widoczny dla wszystkich użytkowników w sieci. Teraz możemy przejść do tworzenia treści dla naszej strony.

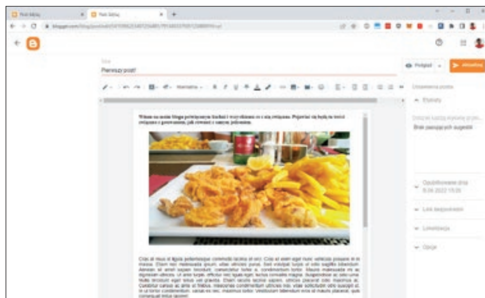
Dodajemy wpisy do naszego bloga

Zanim przejdziemy do zmiany układu lub motywu naszego bloga, najlepiej jest dodać kilka wpisów. Mogą być to dowolne treści, nawet niezawierające żadnych istotnych treści. Pierwsze wpisy posłużą nam do przetestowania różnego typu układów oraz motywów w późniejszym etapie. Dzięki takim próbnym postom będziemy mogli zdecydować się, jak finalnie wizualnie ma wyglądać nasz blog i zaczniemy tworzyć właściwe posty.

1 Po przejściu do panelu zarządzania blogami wybieramy nasz blog, a następnie klikamy po lewej stronie na **Nowy post**.

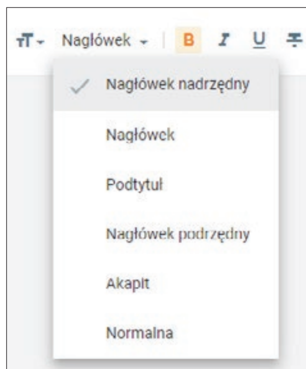


2 Uruchomiony zostanie edytor postów. Wyglądem bardzo przypomina edytory tekstu takie jak Microsoft Office lub inne

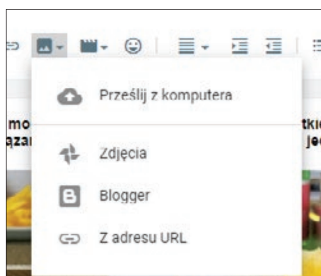


tego typu. Dzięki temu praktycznie każdy użytkownik będzie mógł bez problemu tworzyć treść.

3 Pamiętajmy, aby zawsze dodawać tytuł naszego postu – dzięki temu łatwo będzie później nimi zarządzać. Wprowadzając treść bloga, korzystajmy z różnych stylów, na przykład **Nagłówek**, **Akapit**, **Podtytuł** – łatwiej będzie nam formatować tekst.

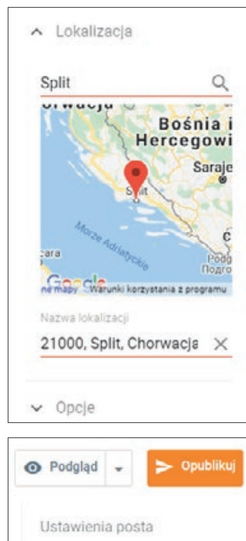


4 Do postu możemy dodawać również obrazy oraz filmy wideo. W celu dodania obrazu klikamy na pasku narzędziowym na ikonę obrazu, a następnie wybieramy źródło zdjęcia.



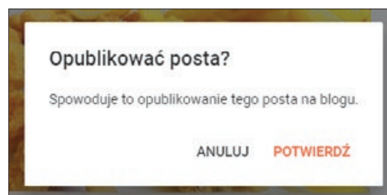
5 Jeśli mamy bibliotekę zdjęć w usłudze Google Zdjęcia, będziemy mogli szybko wstawiać zdjęcia na bloga.

6 Po prawej stronie możemy, korzystając z funkcji **Lokalizacje**, wskazać miejsce związane z tworzonym przez nas postem. Jest to istotne, zwłaszcza gdy tworzymy post związany z jakąś konkretną lokalizacją.

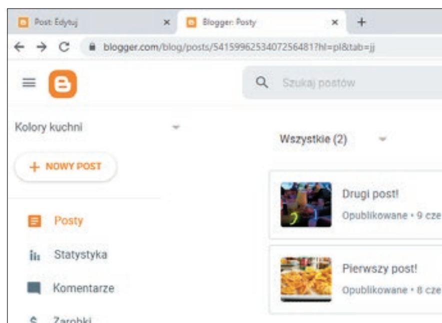


7 Gdy już zakończymy edytowanie postu, klikamy w prawym górnym rogu na **Opublikuj**. Jeśli nie jesteśmy pewni, czy post nie ma błędów, nie musimy się martwić, gdyż w każdej chwili będziemy mogli wyedytować jego treść.

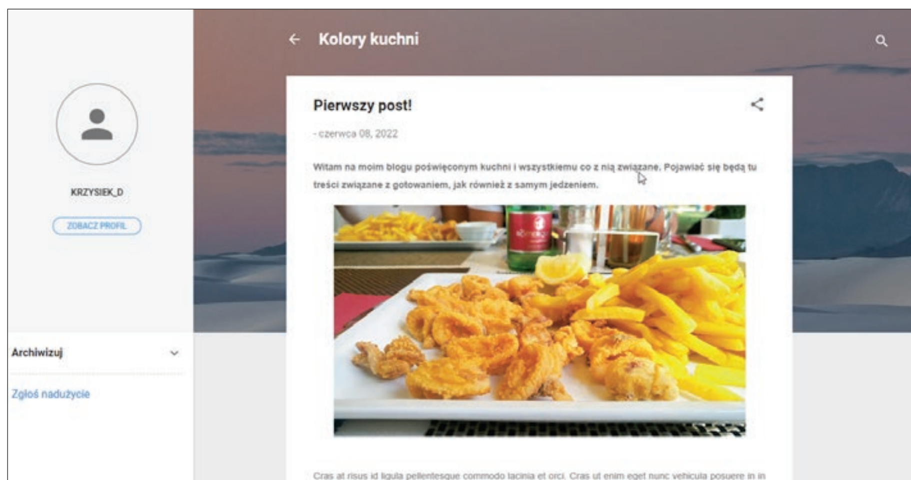
8 Następnie klikamy na **Potwierdź**.



9 Po chwili nasz post zostanie opublikowany i będzie widoczny w oknie zarządzania blogiem. Klikając na niego w tym oknie, od razu przejdziemy do ponownej edycji.



10 Jeśli chcemy przejść do naszego postu i sprawdzić, jak się prezentuje dla każdego czytelnika, wystarczy, że wejdziemy na adres naszego bloga, a następnie kliknie-

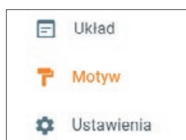


my na **Czytaj więcej** przy naszym nowo dodanym poście.

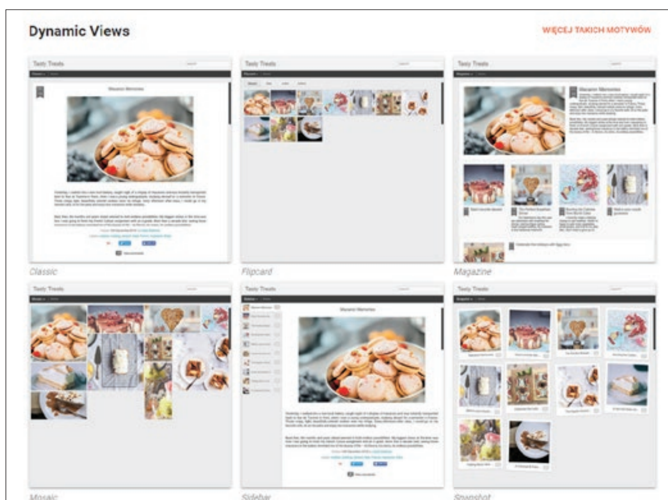
Zarządzamy motywem naszego bloga

Domyślnie sporą część bloga zajmuje przestrzeń z profilem użytkownika zlokalizowana po lewej stronie. Jest to spowodowane wyborem konkretnego motywu dla bloga. Możemy pozbyć się tego paska poprzez zmianę motywu lub zarządzanie układem. Dodatkowo, zmieniając motyw, możemy dopasować go bardziej do tworzonych przez nas treści i w ten sposób sprawić, że czytelnicy będą lepiej odbierali tworzone przez nas treści.

1 Przechodzimy do panelu zarządzania naszym blogiem i po lewej stronie klikamy na **Motyw**.



2 Następnie po prawej przewijamy do motywu, który nam odpowiada, i klikamy na niego. Alternatywnie możemy kliknąć na **Więcej takich motywów**, aby sprawdzić więcej motywów danego typu.



OPCJONALNE KOMENTARZE

Możemy umożliwić czytelnikom bloga pisanie komentarzy do naszych postów. Wystarczy po prawej stronie w polu Opcje wybrać Zezwalaj przy pozycji Komentarze czytelników.

Opcje
Komentarze czytelników
Zezwalaj
Nie zezwalaj, pokazuj istniejące
Nie zezwalaj, ukrywaj istniejące



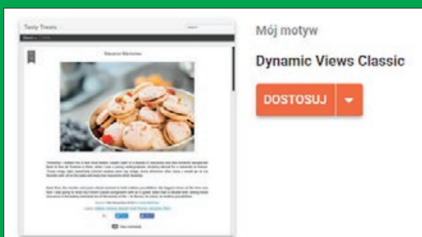
3 Po wybraniu konkretnego motywu możemy kliknąć na **Podgląd** w celu sprawdzenia, jak nasz blog wyglądałby z danym motywem. Jeśli nam się podoba, klikamy na polecenie **Zastosuj**.

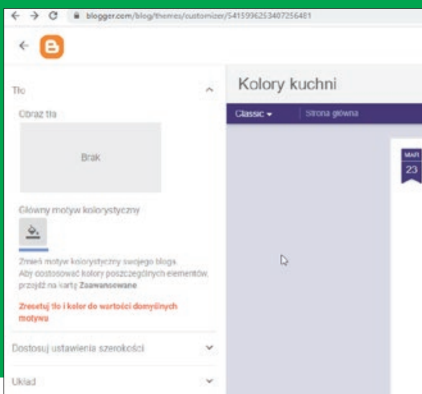
4 Potwierdzamy wybór nowego motywu, klikając na **OK**.



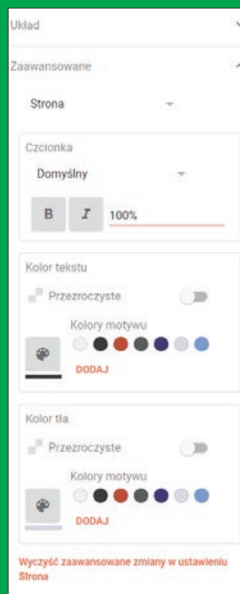
DOSTOSOWYWANIE MOTYWU

Jeśli chcemy jeszcze bardziej dopasować motyw do naszych potrzeb, po wybraniu po prawej stronie w panelu zarządzania pola **Motyw** po lewej przy naszym motywie klikamy na **Dostosuj**.





Możemy w tym oknie wybrać obraz tła lub główny motyw kolorystyczny. Żeby dokonać zmiany, wystarczy kliknąć na jedną z opcji, a następnie wskazać na przykład kolor.



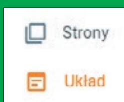
Możemy również wprowadzać bardziej zaawansowane zmiany dotyczące motywu, które będą miały wpływ na czcionkę, kolor tekstu, kolor tła i wiele innych parametrów. W celu zapisania zmian w naszym motywie należy w dolnym prawym rogu kliknąć na ikonę zapisywania.



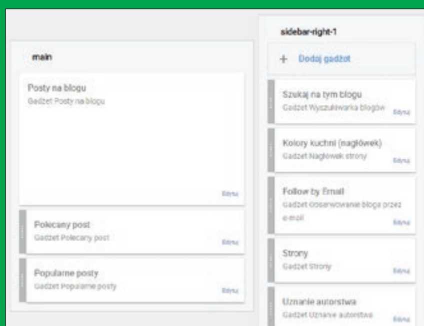
UKŁAD STRONY

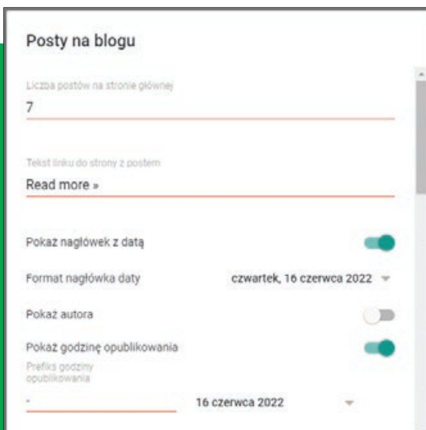
W panelu zarządzania możemy również przejść do edycji układu danego motywu. Jest on różny dla różnych motywów. Dlatego warto najpierw wybrać odpowiadający nam motyw, a dopiero potem popracować nad jego układem.

1 W celu zmiany układu należy po lewej stronie panelu zarządzania kliknąć na **Układ**.



2 Następnie po prawej stronie pojawi się okno z różnego rodzaju prostokątami, które nazywane są gadżetami. Odpowiadają one za pola tekstowe i interaktywne na naszym blogu. Możemy je dowolnie przenosić, przytrzymując i przeciągając w nowe miejsca. Dodatkowo możemy również edytować konkretne gadżety.





3 Edytując gadżet **Posty na blogu**, możemy na przykład zdecydować, ile postów będzie wyświetlanych na stronie głównej, ile komentarzy będzie widocznych, czy będą widoczne przyciski udostępniania, a nawet czy między postami mają być wyświetlane reklamy (musimy wcześniej przeprowadzić konfigurację w karcie **Zarobki**).

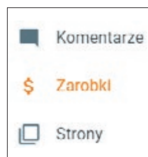
4 Po zakończeniu edycji konkretnego gadżetu należy przewinąć widok do samego dołu okna i kliknąć na **Zapisz**.

ANULUJ ZAPISZ

Aktywujemy reklamy dla bloga

W przypadku korzystania z serwisu **Blogger** dodawanie możliwości wyświetlania reklam jest bardzo proste. Wystarczy przejść do zakładki **Zarobki** i przeprowadzić konfigurację. Należy jednak pamiętać, że zbyt duża ilość reklam może odstraszać naszych czytelników lub uniemożliwiać im komfortowe czytanie tworzonych przez nas treści. Należy zachować balans pomiędzy treścią a ilością reklam. Bardzo aktywne blogi pozwalają na uzyskiwanie stabilnych dochodów co miesiąc. Możemy więc, tworząc bloga, dzielić się naszymi pasjami z innymi, a jednocześnie na tym zarabiać.

1 W panelu zarządzania przechodzimy do zakładki **Zarobki**.



2 Następnie po prawej stronie klikamy na **Utwórz konto AdSense**.



DLA KOGO WŁASNY BLOG

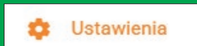
Opisany tu proces tworzenia własnego bloga jest świetną propozycją dla użytkowników, którym brakuje wiedzy i chęci do nauki programowania, gdyż mogą w chwilę rozpocząć tworzenie swojego bloga. Jednak utworzona w tym serwisie strona zawsze będzie jedynie blogiem.

Jeśli chcemy utworzyć bardziej złożoną witrynę, warto skorzystać z bardziej zaawansowanych narzędzi, na przykład WordPressa. W kolejnym rozdziale przeczytamy, co możemy z nim zdziałać i jak to zrobić krok po kroku na naszym komputerze, nie wydając nawet złotówki.

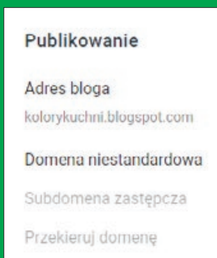
NAZWA DOMENOWA A NASZA WŁASNA STRONA

Korzystając z gotowych rozwiązań typu **Blogger**, musimy liczyć się z tym, że nasz blog będzie zlokalizowany na subdomenie. Oznacza to, że jeśli chcielibyśmy, aby nasz blog miał adres kolorykuchni.com, to po skorzystaniu z Bloggera będzie miał adres kolorykuchni.blogspot.com.
Jeżeli zależy nam na prywatnej domenie, musimy zapłacić jednej ze stron hostingowych. Możemy to również zrobić bezpośrednio z serwisu **Blogger**.

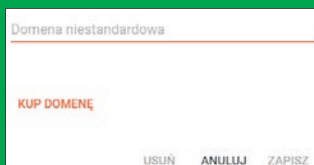
1 Po lewej stronie w panelu zarządzania blogiem klikamy na **Ustawienia**.



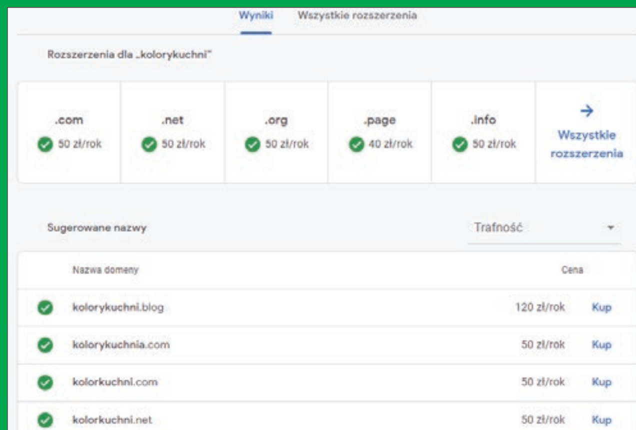
2 Teraz po prawej stronie przewijamy widok do pola **Publikowanie** i klikamy na **Domena niestandardowa**.



3 Jeżeli mamy już zakupioną domenę, możemy przekierować na nią naszego bloga, jeśli nie – klikamy na **Kup domenę**.



4 Zostaniemy przeniesieni do serwisu **Google Domains**, gdzie możemy kupić odpowiadającą nam domenę. Wystarczy wybrać adres oraz rozszerzenie i wnieść opłatę za rok z góry.



3 Wypełniamy formularz rejestracyjny i klikamy na dole strony na **Utwórz konto**. Pamiętajmy o wyborze odpowiedniego kraju dla naszych odbiorców – dzięki temu reklamy będą lepiej dostosowane.

Google AdSense | Rejestracja

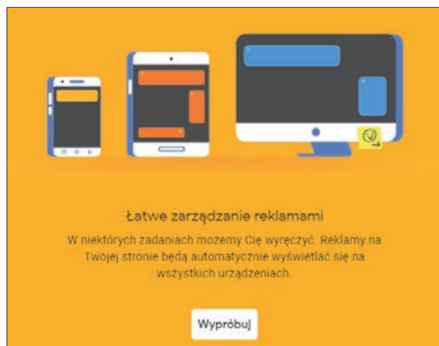
Krzysiek Dziedzic
krzysiek

Twoja witryna
URL Twojej strony
kolorykuchni.blogspot.com

Lepiej wykorzystaj możliwości AdSense
Tak, chcę otrzymywać spersonalizowaną pomoc oraz sugestie dotyczące poprawy skuteczności
Nie, nie chcę otrzymywać spersonalizowanej pomocy ani sugestii dotyczących poprawy skuteczności

Wybór kraju lub terytorium
Kraj lub terytorium
Polska

4 Jeśli wyraziliśmy zgodę na spersonalizowaną pomoc, AdSense samo będzie zarządzać reklamami i umożliwi ich wyświetlanie na wszystkich urządzeniach. Po zapoznaniu się z informacjami klikamy na **Wypróbuj**.



5 Następnie podajemy adres i tworzymy profil płatności, a potem czekamy na zakończenie automatycznej konfiguracji.

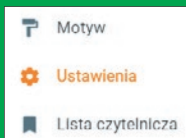
Pracujemy nad konfiguracją
Zwykle sprawdzamy witrynę w kilka dni, czasami jednak
Wtedy możesz zacząć zarabiać na wyświetlaniu reklam.

Uwaga! Taka konfiguracja może potrwać nawet do kilku dni.

BEZPIECZNY BLOG

Jeśli zależy nam na bezpieczeństwie naszym i naszych użytkowników, koniecznie aktywujmy opcję przekierowania HTTPS. Dzięki temu jeśli ktoś będzie chciał odwiedzić naszą witrynę poprzez link z HTTP, co może narazić go na wyciek danych, od razu przekierujemy go na szyfrowaną wersję HTTPS.

1 W panelu zarządzania blogiem po lewej stronie klikamy na **Ustawienia**.



2 Następnie po prawej stronie w opcji **HTTPS** aktywujemy opcję **Przekierowanie HTTPS**.

HTTPS
Przekierowanie HTTPS
Upewnij się, że wizyty w witrynach HTTP są przekierowywane do HTTPS



3 Korzystamy z WordPressa i tworzymy własne strony

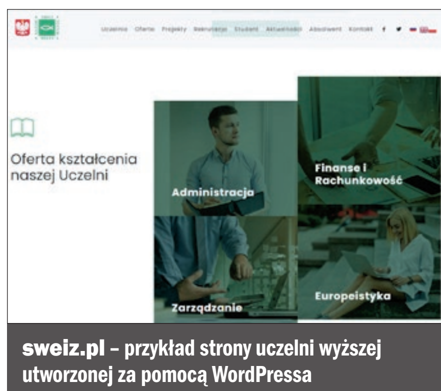
W tym rozdziale poznamy bardzo rozbudowane narzędzie – WordPress – i dowiemy się, jakie są jego możliwości w zakresie tworzenia witryn internetowych. Dzięki lokalnej instalacji całego środowiska będziemy mogli testować różne rozwiązania, nie ponosząc żadnych kosztów

Lokalny serwer

Zdarza się, że osoby, które chcą nauczyć się tworzyć strony internetowe lub aplikacje webowe, zaczynają od wykupienia domeny, hostingu i innych dodatkowych usług. To trochę ryzykowne, bo jeśli stracą zainteresowanie nauką, opłaty i tak będą musiały regularnie wnosić – koszty ponosi się, nawet jeżeli strona nie działa całkowicie poprawnie lub nie jest skończona. Dlatego warto rozważyć na początek tworzenie stron na serwerze



sleepconcept.pl – strona oparta na WordPressie, dobry przykład spójnej wizualizacji



sweiz.pl – przykład strony uczelni wyższej utworzonej za pomocą WordPressa

lokalnym. Funkcję takiego serwera może pełnić po prostu nasz komputer.
Serwer to urządzenie, które jest w stanie udostępniać usługi na potrzeby innych komputerów lub programów. Uruchomienie serwera na laptopie lub komputerze stacjonarnym jest bardzo częstym rozwiązaniem i świetnie się sprawdza do nauki. Największą zaletą postawienia własnego serwera lokalnie jest możliwość testowania stron internetowych bez konieczności umieszczania ich w sieci, a tym samym ponoszenia jakichkolwiek opłat.
Oto kilka argumentów przemawiających za własnym serwerem na witryny WWW:

■ Możliwość testowania stron

Podczas pracy z własnym serwerem możemy sami tworzyć stronę internetową od podstaw. Nie musi być ona opublikowana na naszej domenie i możemy ją dopracować, zanim umieścimy ją w sieci. Mamy więc dowolnie dużo czasu, aby zająć się wszystkimi szczegółami.

■ Edytowanie działających stron

Często zdarza się, że trzeba dokonać zmian na stronie WWW lub w aplikacji sieciowej. Powodem może być rozbudowa lub usunięcie jakichś elementów. Lokalny serwer pozwala na stworzenie środowiska testowego. Podczas gdy tak zwane środowisko produkcyjne będzie aktywne i dostępne w sieci dla użytkowników, my – offline – możemy dokonać wszelkich potrzebnych zmian, a następnie podmienić pliki na serwerze umieszczonym w sieci.

MINUSY SERWERA LOKALNEGO

Serwer lokalny na laptopie czy pececie sprawdza się do nauki i eksperymentowania, ale nie nadaje się do hostowania profesjonalnej strony internetowej. Takie urządzenie musiałoby działać bez przerwy całą dobę, a dodatkowo, jeśli nasza strona miałaby dużo odwiedzających, mogłoby się okazać, że nie będzie odpowiednio szybko ładowana lub nawet zostanie zawieszony serwer przez zbyt duży ruch. Jeśli chcielibyśmy kupić i złożyć serwer z prawdziwego zdarzenia, musielibyśmy wydać bardzo dużo pieniędzy, co jest całkowicie nieopłacalne – znacznie lepiej skorzystać z dostępnych hostingów online.

■ Wygodne kopie zapasowe

Każda zmiana na stronie internetowej powinna być poprzedzona wykonaniem kopii zapasowej. Firmy hostingowe oferują takie usługi często za dodatkową opłatą. W przypadku własnego lokalnego serwera możemy wykonać kopie zapasowe ręcznie, a następnie sprawdzić lokalnie, czy kopia została poprawnie utworzona, co jest bardzo ważne. Gdyby pliki kopii były tworzone niepoprawnie, w przypadku awarii nie moglibyśmy przywrócić naszej strony.

■ Sprawdzanie nowych rozwiązań

W przypadku WordPressa, jak również innych systemów CMS, mamy możliwość instalowania wtyczek, które mogą znacząco



albertzawada.com – strona znanego fotografa utworzona w WordPressie. Takie strony są dobrym rozwiązaniem dla freelancerów



bini.co/pl – przykład sklepu internetowego utworzonego w WordPressie

rozszerzać funkcjonalność naszej strony lub zmieniać jej wygląd. Niestety, zdarza się, że dodatek, który na stronie autora prezentuje się imponująco, po zainstalowaniu na naszej witrynie nie działa lub powoduje błędy. Serwer lokalny to dobre rozwiązanie do testowania różnego typu nowości, gdyż zmiany wprowadzane offline nie mają znaczenia dla użytkowników, a my możemy eksperymentować i rozwijać naszą witrynę.

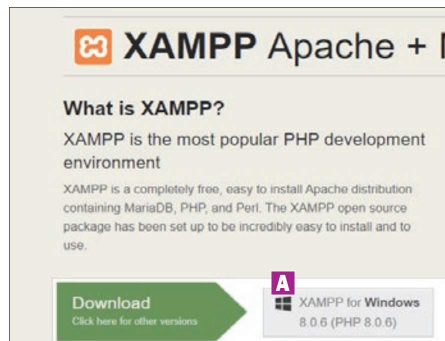
WordPress na własnym komputerze

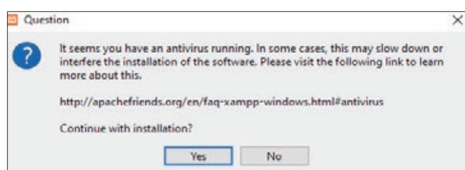
W tej części rozdziału zobaczymy, jak uruchomić lokalnie serwer z **WordPressem** (DVD-KOD: 035). Możemy wszystko wykonywać od podstaw sami, co jest jednak wskazane dla bardziej zaawansowanych użytkowników. W przypadku użytkowników początkujących zalecane jest skorzystanie z gotowego oprogramowania, jak na przykład **XAMPP** (DVD-KOD: 036).

1 Instalujemy XAMPP

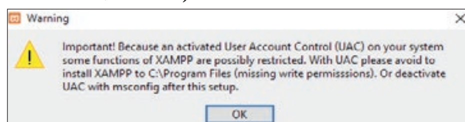
XAMPP to zestaw oprogramowania, który pozwala na uruchamianie lokalnie stron internetowych. Składa się na niego oprogramowanie **Apache**, **phpMyAdmin**, baza danych **MariaDB**, **PHP** oraz **Perl**. Program doskonale działa w środowisku Windows. Możemy go zainstalować z płyty dołączonej do książki lub ze strony **apachefriends.org/index.html** (w tym wypadku na stronie klikamy na kafelek z logo Windows **A** w celu pobrania instalatora).

1 Po uruchomieniu instalatora pojawi się informacja o możliwych problemach z oprogramowaniem antywirusowym – wyłączamy osłony na czas instalacji i klikamy na **Yes**.

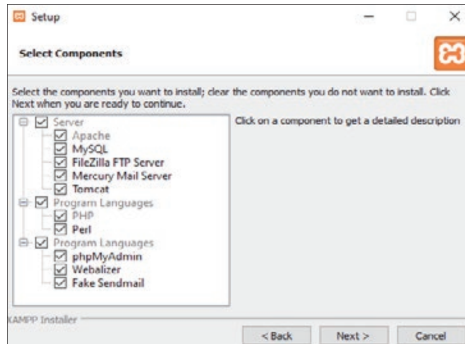




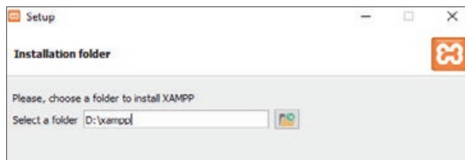
2 Następnie pojawi się ostrzeżenie dotyczące uprawnień systemu Windows. Nie należy instalować XAMPP na partycji **C** – jeśli mamy taką możliwość, lepiej zainstalować ten program na innym dysku. Jeżeli nie mamy takiej możliwości, trzeba dezaktywować **UAC**, czyli zabezpieczenia systemu Windows, co nie jest zalecane.



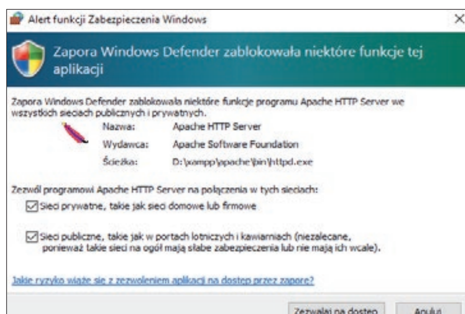
3 Następnie rozpoczynamy instalację – zaznaczamy wszystkie komponenty do zainstalowania i klikamy na **Next**.



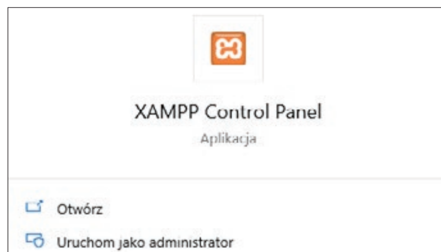
4 Wskazujemy lokalizację docelową instalacji i klikamy na **Next**.



5 W kolejnym oknie wybieramy język angielski, usuwamy zaznaczenie przy opcji **Learn more about Bitnami for XAMPP** i klikamy na **Next**. Po przejściu do kolejnego okna rozpocznie się instalacja, która trwa dość długo. W trakcie instalacji wyrażamy zgodę na dostęp do sieci dla serwera Apache.



6 Po zakończeniu instalacji zaznaczamy opcję **Do you want to start the Control Panel now?** i klikamy na **Finish**.

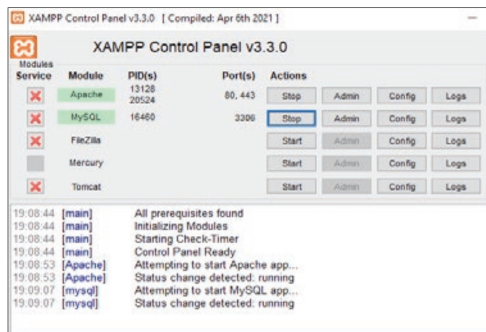


2 Pierwsze kroki i sprawdzenie instalacji

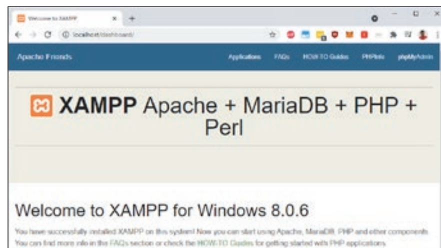
Po zainstalowaniu XAMPP musimy pamiętać, że serwer tak naprawdę udostępnia usługi dla użytkowników. Musimy go uruchomić oraz inne dodatkowe usługi. Na początek będzie potrzebny nam tylko serwer Apache oraz baza danych MySQL (MariaDB). Uruchamianie tych komponentów najwygodniej przeprowadzić w panelu kontrolnym XAMPP.

Panel kontrolny po raz pierwszy uruchomimy, kończąc instalację, później wystarczy wpisać frazę XAMPP Control Panel w pole wyszukiwania Windows i uruchomić aplikację.

1 W panelu kontrolnym XAMPP klikamy na **Start** w wierszach **Apache** oraz **MySQL**.



2 Jeżeli do tej pory wszystkie kroki wykonaliśmy poprawnie, uruchamiamy przeglądarkę internetową, w pasku adresowym wpisujemy **localhost** i wciskamy enter. Jeśli pojawi się strona z informacjami o XAMPP, serwer działa poprawnie.



3 Już na tym etapie mamy aktywny serwer PHP, który działa lokalnie i wyświetla

stronę WWW. Jeśli zamierzamy wyświetlać naszą stronę, musimy wykonać znacznie więcej kroków.

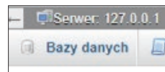
Uwaga! XAMPP Control Panel uruchamiamy zawsze jako administrator. Jeżeli po uruchomieniu usługi MySQL nie pojawi się port, wyłączamy program i uruchamiamy go ponownie jako administrator.

3 Tworzymy nową bazę danych

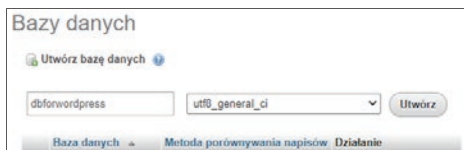
Jeśli zamierzamy korzystać z WordPressa w środowisku lokalnym, musimy utworzyć nową bazę danych, która będzie zawierać wszystkie informacje naszej strony internetowej – komentarze, loginy, hasła, posty, ustawienia itp. Domyślną bazą dla XAMPP jest MariaDB. Uruchomiliśmy już moduł bazy w panelu kontrolnym, teraz możemy przejść do tworzenia nowej bazy.

1 Bazę w **XAMPP** zakłada się z panelu **phpMyAdmin** **A** – w przeglądarce wpisujemy adres **localhost/phpmyadmin** i wciskamy enter.

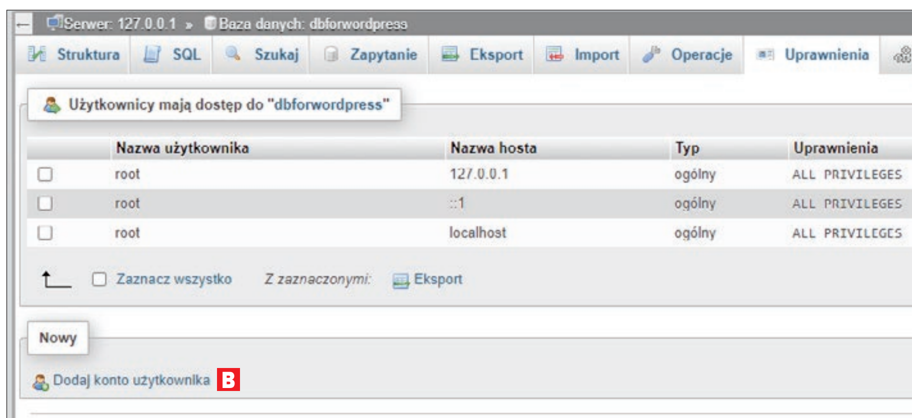
2 Teraz na górnym pasku klikamy na **Bazy danych**.

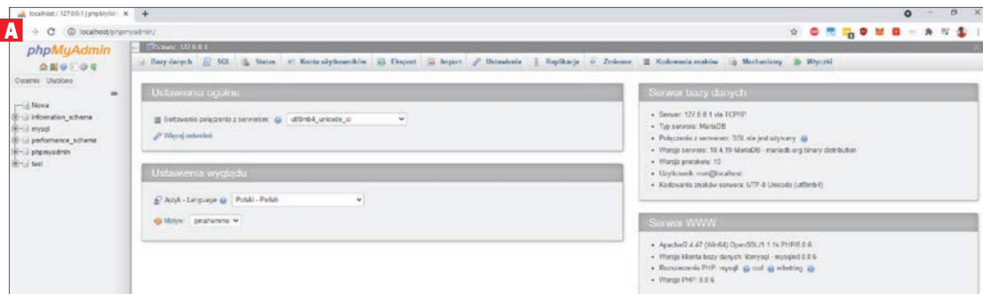


3 Tworzymy nową bazę danych, na przykład **dbforwordpress**, wybieramy metodę porównywania napisów **utf8_general_ci** i klikamy na **Utwórz**.



4 Teraz dodamy użytkownika naszej bazy – klikamy na górnym pasku na zakładkę **Uprawnienia**, a następnie na **Dodaj konto użytkownika** **B** w polu **Nowy**.

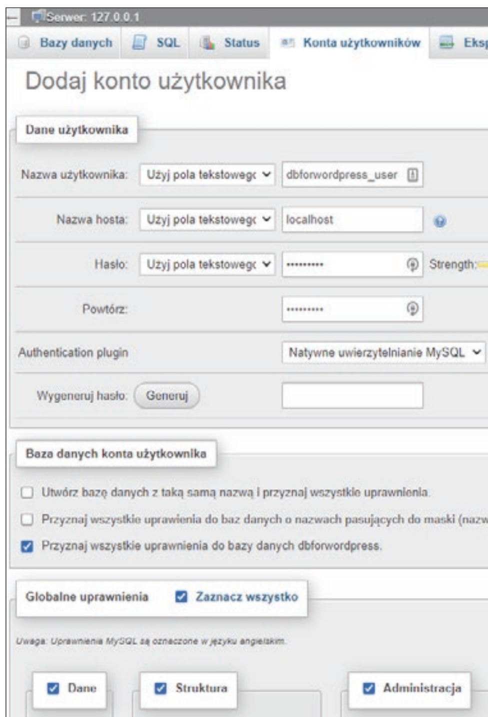






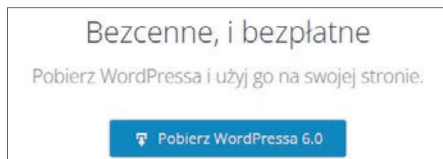
5 Podajemy nazwę użytkownika, host, hasło, zaznaczamy opcję **Zaznacz wszystko** w polu **Globalne uprawnienia** i klikamy na **Wykonaj** u dołu strony.

6 Po utworzeniu pojawi się wpis **C**, który zawiera pełną instrukcję tego, co „wyklikaliśmy” oraz wpisaliśmy w formularzu.



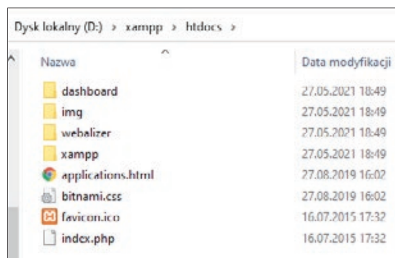
4 Lokalna instalacja WordPressa

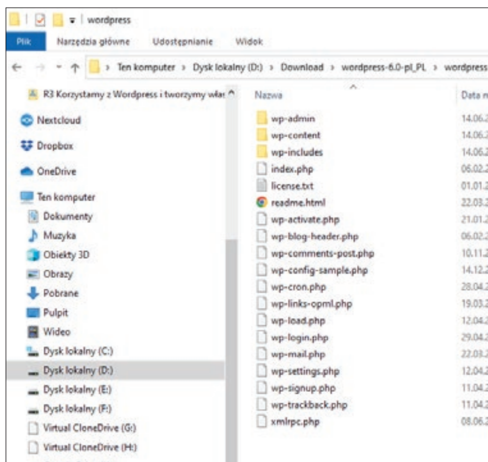
Po uruchomieniu serwera WWW i stworzeniu bazy danych z użytkownikiem pobieramy WordPressa z oficjalnej strony – **wordpress.org/download** (możemy go też zainstalować z płyty dołączonej do książki).



1 Po pobraniu wypakowujemy WordPressa z archiwum.

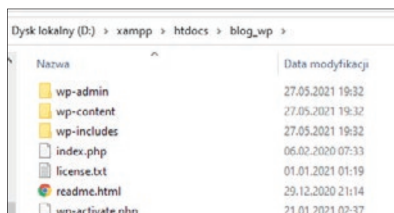
2 Teraz przechodzimy do lokalizacji, w której zainstalowaliśmy **XAMPP**, i wchodzimy do folderu **htdocs**.



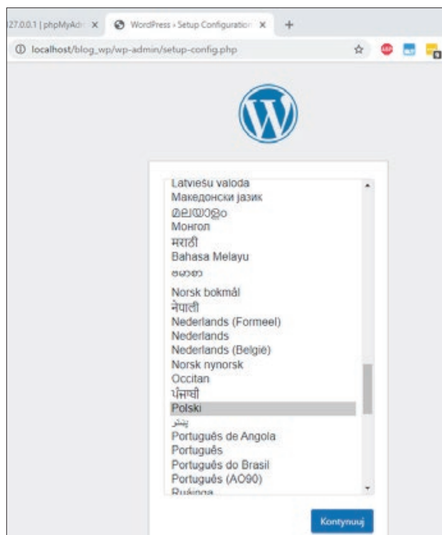


3 Usuwamy z niego wszystkie pliki i foldery. Tworzymy nowy folder o nazwie na przykład **blog_wp** i wklejamy do niego

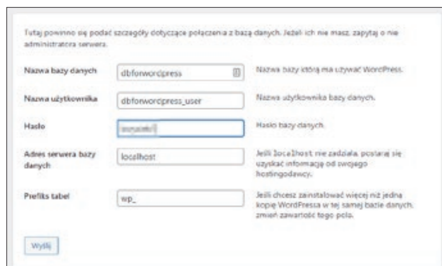
zawartość całego wypakowanego folderu **wordpress** z pobranego archiwum.



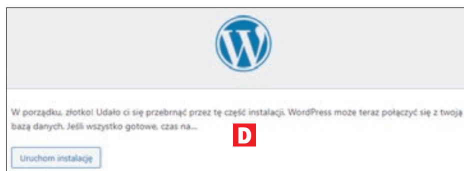
4 Teraz w przeglądarce wpisujemy **localhost/blog_wp**, aby rozpocząć proces instalacji WordPressa. W pierwszym oknie wybieramy język **Polski** i klikamy na **Kontynuuj**.



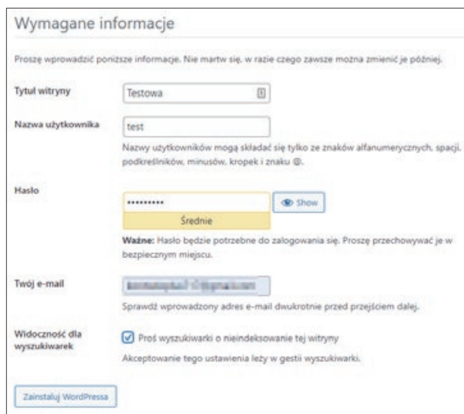
5 W kolejnym oknie klikamy na **Zaczynajmy** i rozpoczynamy konfigurowanie połączenia z bazą danych. Podajemy nazwę bazy, użytkownika oraz hasło, wskazujemy adres, zostawiamy domyślny prefiks i klikamy na **Wyślij**.



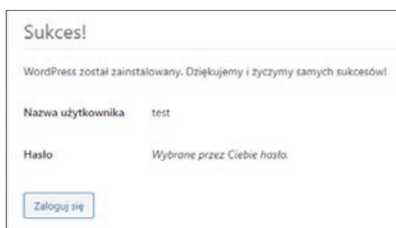
6 Jeśli pojawi się nam taka **D** informacja, oznacza to, że do tej pory poprawnie wykonaliśmy wszystkie kroki. Klikamy na **Uruchom instalację**.



7 Teraz podajemy tytuł witryny, nazwę użytkownika, jego hasło, adres e-mail, zaznaczamy opcję **Proś wyszukiwarki o nieindeksowanie tej witryny** i klikamy na **Zainstaluj WordPressa**. Proces instalacji trwa około pięciu minut.



8 Po zakończeniu instalacji klikamy na **Zaloguj się**.



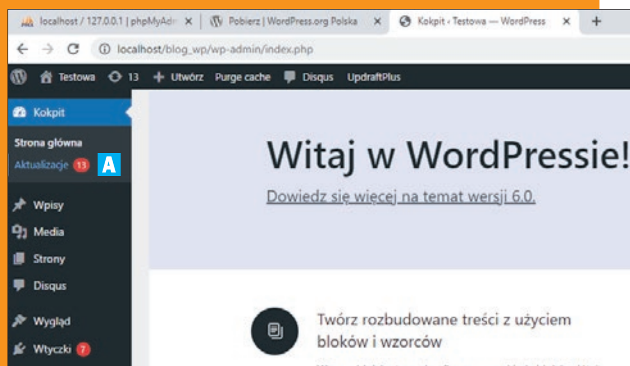
AKTUALIZACJA WORDPRESSA

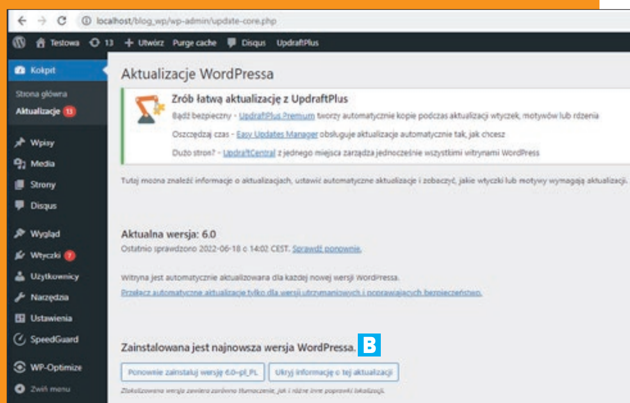
Warto pamiętać o częstym aktualizowaniu całego modułu WordPressa, gdyż dziury w zabezpieczeniach lub błędy oprogramowania mogą sprawić, że nasza witryna nie będzie bezpieczna dla użytkowników lub nie będzie poprawnie wyświetlana. Proces aktualizacji jest dość prosty, jednak zaleca się ręczne wykonanie kopii zapasowej całego folderu z wszystkimi plikami i wtyczkami.

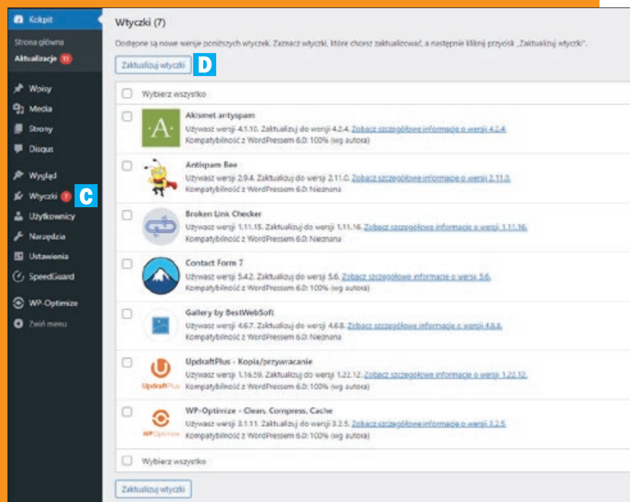
1 Po zalogowaniu w panelu administracyjnym WordPressa klikamy po lewej stronie na **Aktualizacje** **A**.

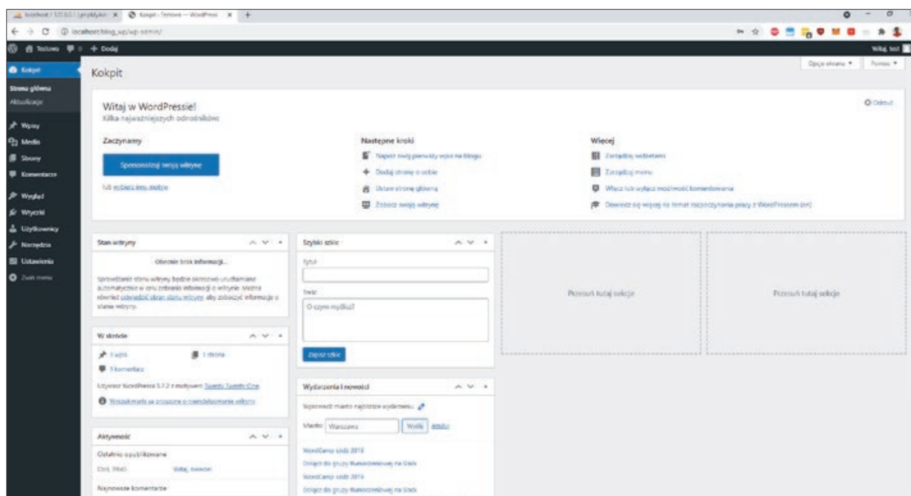
2 Następnie, jeśli nie mamy zainstalowanej najnowszej wersji WordPressa **B**, klikamy na instalację najnowszej wersji. Domyślnie aktualizacje instalowane są automatycznie przynajmniej raz na dobę, jeśli serwer jest aktywny.

3 Musimy również pamiętać w miarę dodawania różnego rodzaju wtyczek, że ich aktualizowanie musimy wykonywać zawsze ręcznie. Dlatego też przewijamy widok w dół do sekcji **Wtyczki** **C** i klikamy na **Zaktualizuj wtyczki** **D**. W ten sposób będziemy mogli dokonać aktualizacji wszystkich dostępnych wtyczek.









9 Podajemy dane logowania i ponownie klikamy na **Zaloguj się**.

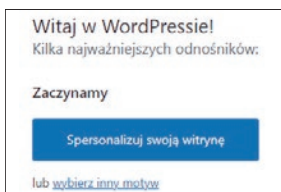
10 Po zalogowaniu pojawi się panel administratora WordPressa dla naszej strony. To właśnie z jego poziomu będziemy mogli zarządzać naszą stroną. Domyślnym motywem po instalacji jest **Twenty Twenty-One**.

Edytujemy domyślną stronę

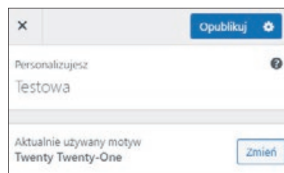
Po wykonaniu powyższych instrukcji będziemy mieć działający lokalny serwer z bazą danych oraz aktualnego WordPressa z gotową domyślną witryną. Możemy teraz przejść do zmieniania motywu oraz edytowania samej strony lokalnie.

Zmiana motywu

1 Logujemy się do panelu administratora naszej strony – strona logowania to **localhost/[folder_dla_wordpress]/wp-login.php**. Następnie klikamy na **Spersonalizuj swoją witrynę**.



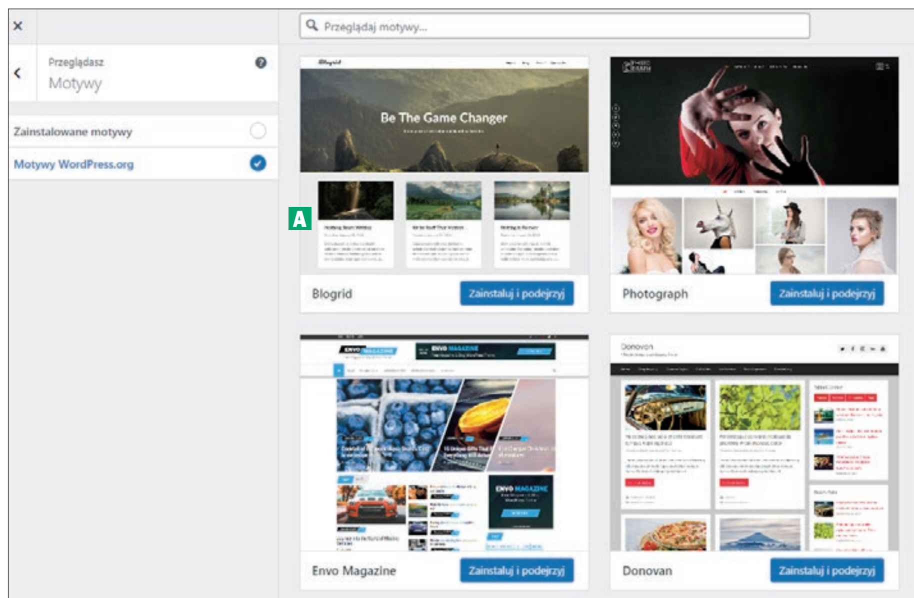
2 Teraz po lewej stronie klikamy na **Zmień** przy polu **Aktualnie używany motyw**.



3 Następnie klikamy na **Motywy WordPress.org** i z okna po prawej stronie wybieramy motyw odpowiedni dla naszej witryny **A**.

4 Po wybraniu motywu klikamy na **Aktywuj i opublikuj** **B**.

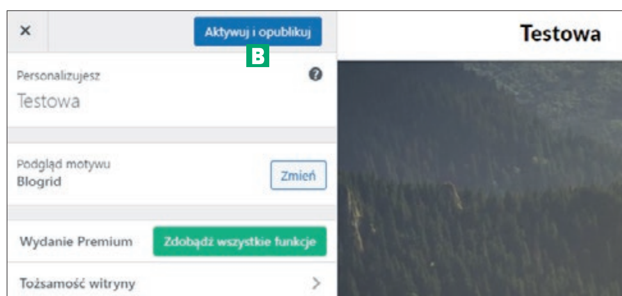
5 Nowy motyw będzie teraz domyślny dla naszej witryny.



Dodajemy nowe strony lub wpisy

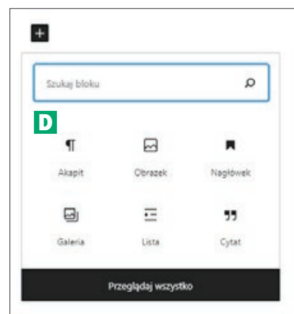
1 Po zalogowaniu do panelu administratora klikamy na **Dodaj strony** lub **Dodaj wpis**.

Następne kroki
- Edytuj stronę startową
- Dodaj strony
- Dodaj wpis
- Zobacz swoją witrynę

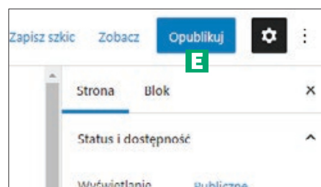


2 Następnie tworzymy nasz wpis, korzystając z bloków. W celu dodania kolejnego bloku klikamy na plus **C** u dołu wpisu i wybieramy rodzaj bloku **D**.

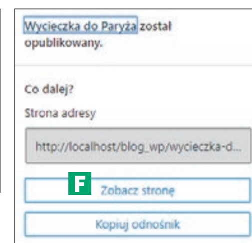




3 Po wprowadzeniu wszystkich treści klikamy w górnym prawym rogu na **Opublikuj** **E**.



4 W celu podejrzenia opublikowanej strony klikamy na **Zobacz stronę** **F**.



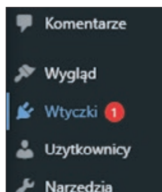
Dodajemy wtyczki do naszego WordPressa

Możliwości instalacji wtyczek mamy kilka w zależności od tego, w jakim środowisku pracujemy. Jeśli korzystamy z lokalnego środowiska, które skonfigurowaliśmy według opisu powyżej – najprościej przekopiować pobrany plik z wtyczką do odpowiedniego katalogu lokalnie na naszym dysku. Poznajmy kilka sposobów, które mogą być przydatne w zależności od konfiguracji naszego serwera.

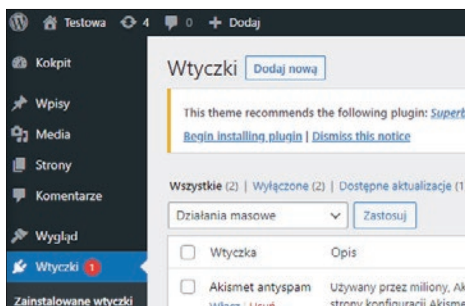
Instalacja automatyczna w panelu administratora

Jest to jedna z najprostszych form instalacji wtyczek w WordPressie. Ogranicza nas nieco, gdyż możemy wybrać do zainstalowania jedynie oficjalne dostępne wtyczki – jednak dla początkujących jest to najbardziej zalecana i najbezpieczniejsza forma instalacji.

1 Logujemy się do panelu administratora naszej witryny WordPress. Następnie w menu po lewej stronie klikamy na **Wtyczki**.

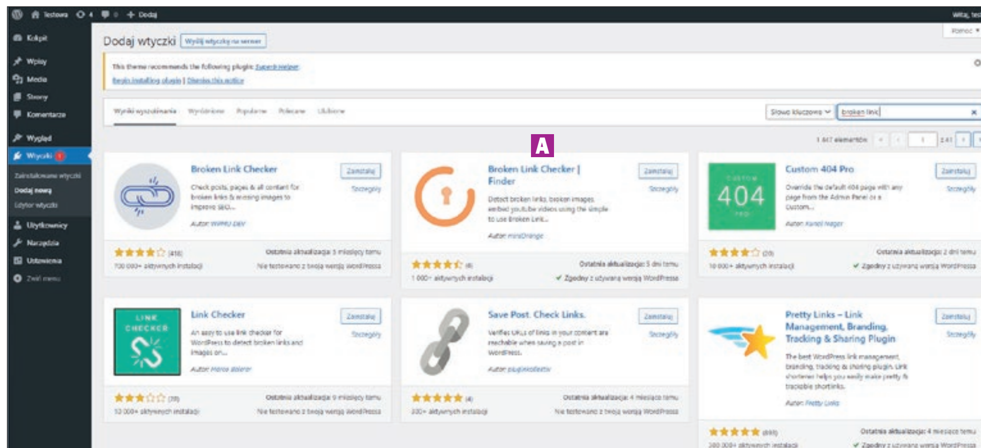


2 Teraz po prawej stronie klikamy w górnej części ekranu na **Dodaj nową**.



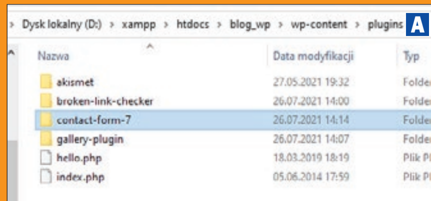
3 Korzystając z listy wyróżnionych wtyczek lub pola wyszukiwania odnajdujemy tę, która będzie dla nas przydatna. Jedną z ciekawszych wtyczek jest **Broken Link Checker** **A**, która sprawdza, czy linki umieszczone na naszych stronach są aktywne.

4 Po kliknięciu na nazwę wtyczki możemy poznać szczegółowe informacje na jej temat **B**, następnie klikamy na **Zainstaluj**.

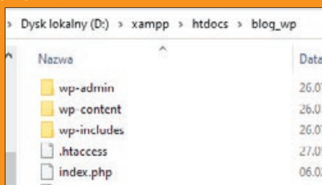


WGRYWANIE WTYCZEK BEZPOŚREDNIO DO KATALOGU WORDPRESSA

Ta metoda jest najwygodniejsza, jeśli mamy wiele wtyczek do przetestowania i robimy to w środowisku lokalnym. Wystarczy rozpakować archiwum z wtyczką i przenieść zawartość do odpowiedniej lokalizacji.

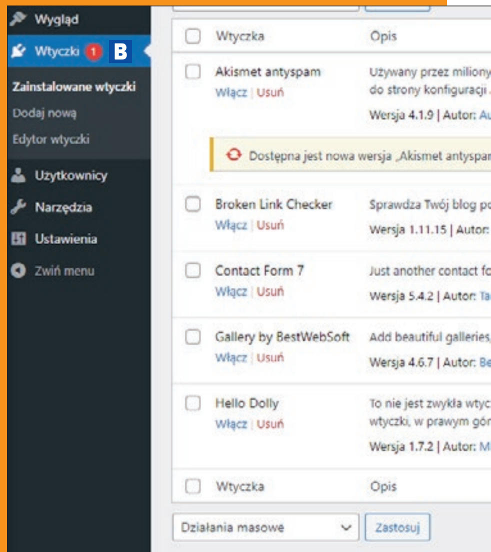


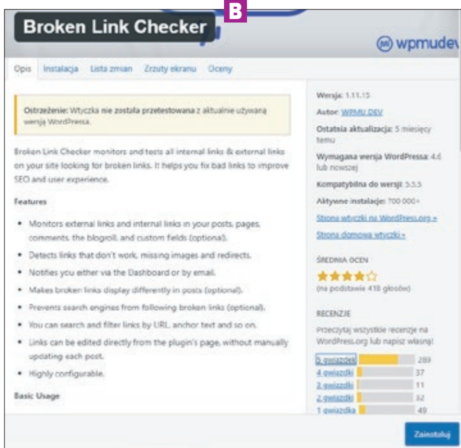
1 Otwieramy Eksplorator i przechodzimy do lokalizacji, w której zainstalowaliśmy XAMPP. Następnie przechodzimy do folderu **htdocs** i folderu głównego naszej witryny.



2 Przechodzimy do folderu **wp-content**, **plugins** **A**, gdzie przenosimy wypakowaną zawartość pobranego archiwum ZIP zawierającego wtyczkę.

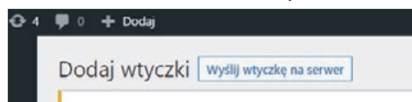
3 Teraz wystarczy jedynie przejść w panelu administratora do zakładki **Wtyczki** **B**, a nasza wtyczka będzie zainstalowana. Pozostaje jedynie jej aktywacja.





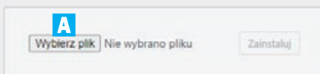
Instalacja pobranych wtyczek w panelu administratora

Część wtyczek dostępna jest tylko na stronach twórców, czasem też w oficjalnej wyszukiwarce nie będziemy w stanie znaleźć starszych wersji wtyczek, które są kompatybilne z naszą wersją WordPressa, dlatego też warto umieć zainstalować wtyczki ręcznie.



1 Logujemy się do panelu administratora WordPressa, klikamy na **Wtyczki**, a następnie na **Dodaj nową**. Po prawej stronie klikamy na **Wyślij wtyczkę na serwer**.

Jeżeli posiadasz wtyczkę w formacie .zip, możesz zainstalować lub aktualizować ją poprzez wysłanie jej na serwer za pośrednictwem tej strony.



2 Teraz klikamy na **Wybierz plik** **A**, wskazujemy archiwum ZIP z wtyczką na naszym dysku i klikamy na **Otwórz**.

3 Po załadowaniu pliku klikamy na **Zainstaluj**.



4 Po chwili wtyczka zostanie zainstalowana i będzie gotowa do włączenia **B**.

Instalacja wtyczki z wysłanego na serwer pliku: gallery-plugin.4.6.7.zip

Rozpakowywanie paczki...

Instalacja wtyczki...

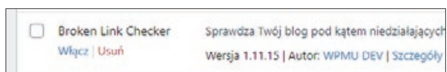
Wtyczka została zainstalowana.

B Włącz wtyczkę | Przejdź do instalatora wtyczek

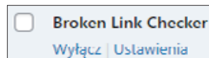
Uruchamiamy wtyczki

Po zainstalowaniu wtyczki będą dostępne w panelu zarządzania, jednak jeśli chcemy z nich korzystać na naszej stronie, musimy je włączyć.

1 Po zainstalowaniu wtyczki w zakładce **Wtyczki**, **Zainstalowane wtyczki** klikamy na **Włącz** przy wtyczce, która ma być aktywowana.



2 W zależności od wtyczki konfiguracja **C** może przebiegać różnie, zwykle należy kliknąć na **Ustawienia**.



Opcje pluginu Link Checker **C**

This theme recommends the following plugin: Superb Helper.
Begin installing plugin | Dismiss this notice

Ogólne | Szukasz linków w | Które linki sprawdzać | Protokoły i API | Zaawansowane

Stan Pokaż informacje typu debug	Nie znaleziono niedziałających linków Brak URLi w kolejce do sprawdzenia. Wykryto 9 unikatowych adresów URL z 9 linków.
Sprawdź każdy link	Co 72 godzin Częstość sprawdzania istniejących linków. Nowe linki będą sprawdzane tak szybko, jak to możliwe.
Powiadomienia na E-mail	☑ Wysyłaj do mnie E-mail z wiadomością o nowowykrytych niedziałających linkach. ☐ Powiadom mailem autora o błędnych odnośnikach we wpisach
Adres e-mail do powiadomień	Pozostaw puste, by użyć adresu ustawionego w Ustawienia → Ogólne.
Control external links	Install the free External Links plugin to control if external links open in a new tab, and their nofollow, noopener and UGC options. It's used on over 80,000 sites just like yours.
Linki-dodatki	☑ Ustaw odmienne formatowanie niedziałających linków | Edytuj CSS ☐ Zastosuje własne formatowanie do niedziałających linków | Edytuj CSS ☐ Nie pozwól wyszukiwarkom podążać po niedziałających linkach Ustawienia dotyczą tylko treści wpisów, a nie komentarzy i własnych pól.
Sugestie	☑ Sugeruj zamienniki do błędnych linków
Ostrzeżenia	☑ Pokaż niepewne lub drobne problemy jako "ostrzeżenia" zamiast "uszkodzone" Wyłączenie tej opcji spowoduje, że wtyczka zgłosi wszystkie problemy jako uszkodzone linki.

3 Rozpoczynamy konfigurację wtyczki do naszych potrzeb. W przypadku wtyczki do sprawdzania nieaktywnych linków, jeśli nasza witryna nie jest zbyt rozbudowana, możemy ustalić czas pomiędzy kolejnymi sprawdzeniami nawet na 24 godziny. Dodatkowo w zakładkach **Szukasz linków** oraz **Które linki sprawdzać** możemy zdefiniować konkretne obiekty naszej strony, jakie mają być sprawdzane, na przykład komentarze czy wpisy, oraz rodzaje linków do weryfikacji.

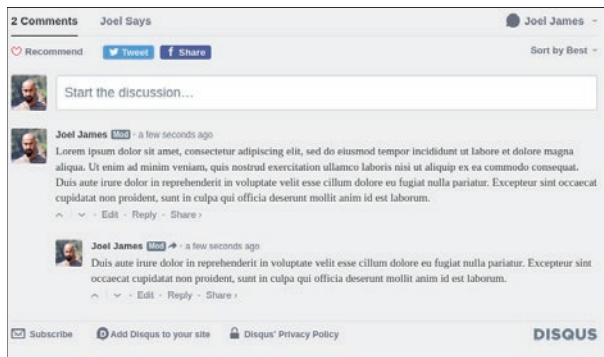
Polecane wtyczki

Poznajmy kilka wtyczek przydatnych dla każdego, kto tworzy swoją stronę, wykorzystując WordPressa.

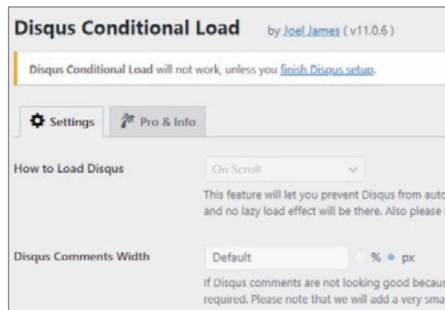
■ **Broken Link Checker** – pozwala sprawdzić, czy wszystkie linki na naszej stronie działają **A**. Dzięki temu, gdy jakiś link wygaśnie, zostaniemy powiadomieni i będziemy mogli go podmienić, a użytkownicy nie odczują problemu związanego z niedziałającymi linkami. Dostęp do listy linków uzyskamy, klikając w panelu administratora na **Narzędzia**, **Niedziałające linki** po aktywowaniu wtyczki.

■ **Disqus Conditional Load** – po zainstalowaniu i aktywowaniu ta wtyczka podmienia domyślny system komentarzy WordPressa. Disqus jest usługą zewnętrzną, która ułatwia komentowanie na stronach internetowych. Bardzo dużą zaletą tego systemu komentarzy jest możliwość posiadania konta, które

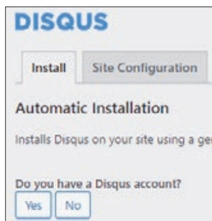
umożliwia komentowanie wielu stron internetowych jednym profilem. Dzięki temu użytkownicy, którzy mają już zarejestrowane konto, będą mogli bez problemu komentować na naszej stronie, a inni użytkownicy będą mogli ich rozpoznać po profilu – sprawia to, że komentowanie staje się mniej anonimowe i można nawiązać konstruktywne dyskusje.



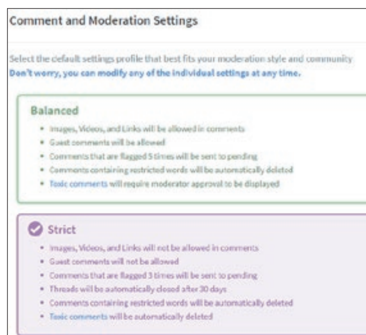
1 Po zainstalowaniu i włączeniu wtyczki przechodzimy do jej ustawień i klikamy na **finish Disqus setup**.



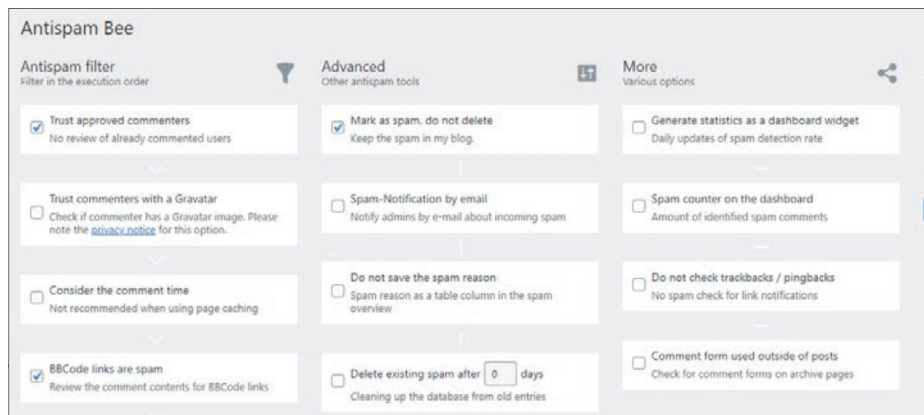
2 Następnie klikamy na **No** i tworzymy konto Disqus.



3 Podczas konfiguracji wybieramy, jaki typ administrowania komentarzami nam odpowiada – zawsze możemy go zmienić.



■ **Antispam Bee** – pozwala na blokowanie spamu w komentarzach. Jeśli nie używamy Disqus, warto skorzystać z tej wtyczki zamiast standardowej Akismet, która nie sprawdza się tak dobrze. Filtrowanie spamu możemy dostosowywać w ustawieniach. Pamiętajmy, że wszelkie zmiany zatwierdzamy, klikając na **Save Changes**.

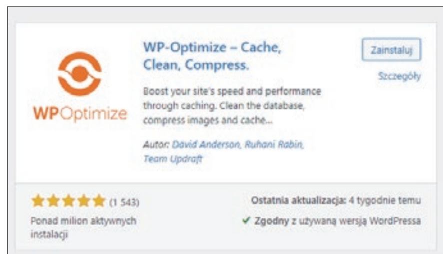


Optymalizujemy naszą stronę

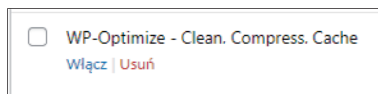
Ponieważ po zainstalowaniu i aktywowaniu wtyczek nasza strona może przestać płynnie się wyświetlać, musimy ciągle dbać o jej optymalizację. Dodatkowo jeśli dodamy zbyt wysokiej jakości zdjęcia bezpośrednio do naszej strony, każdy odwiedzający będzie musiał je załadować, a to znacząco obciąży serwer, na którym jest hostowana, i sprawi, że odwiedzający będą musieli dość długo czekać na załadowanie strony.
Im większa i bardziej rozbudowana strona, tym więcej problemów z jej optymalizacją, ponieważ później pojawiają się problemy ze zbyt rozbudowaną bazą danych, linkami i innymi zmiennymi, o których musimy pamiętać. Dlatego zaleca się, by już od początku tworzenia strony dbać o jej przejrzystość oraz aby nie zaśmiecać zbędnymi wtyczkami i plikami katalogu głównego. Dodatkowo warto korzystać z wtyczki **WP-Optimize**, która służy głównie do poprawy wydajności naszej strony i umożliwia testowanie na przykład czasu ładowania.

Instalujemy WP-Optimize

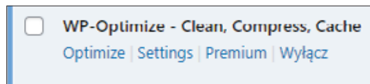
1 Instalujemy wtyczkę **WP-Optimize – Cache, Clean, Compress** w panelu administratora naszej strony.



2 Włączamy wtyczkę po zainstalowaniu.

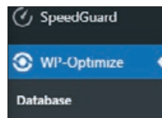


3 Następnie klikamy na **Optimize**.

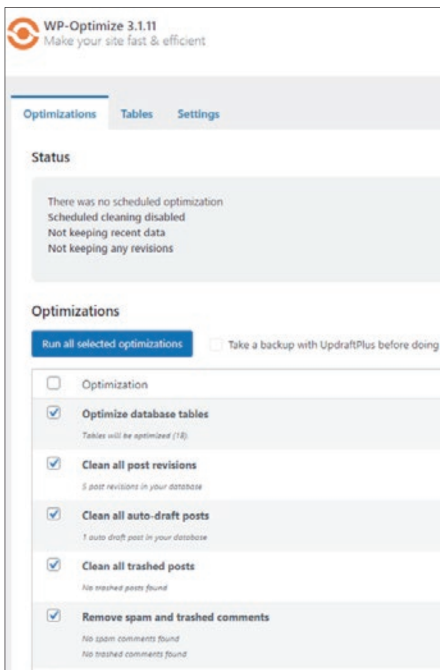


Optymalizujemy bazę danych

1 Po lewej stronie w panelu administratora klikamy na **WP-Optimize**, a potem klikamy na **Database**.

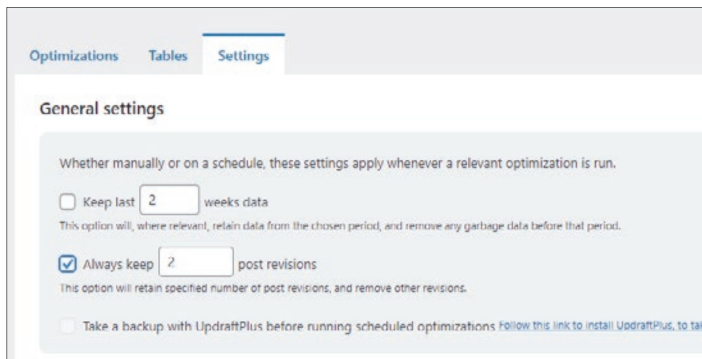


2 Następnie klikamy na **Run all selected optimizations**.



3 W trakcie tego procesu między innymi zostaną usunięte wszystkie posty przeniesione do kosza i zapisane wersje robocze stron, które już są opublikowane, zostanie też zoptymalizowana cała baza danych. Jeśli

często, tworząc nowe posty, zapisujemy je w formie roboczej lub chcemy móc wracać do poprzednich wersji postu, musimy przejść do zakładki **Settings** i zaznaczyć opcję **Always keep 2 post revisions**.



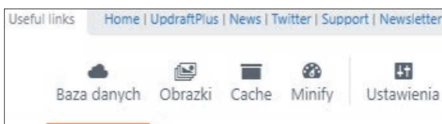
4 W zakładce **Settings** możemy aktywować harmonogram optymalizacji bazy danych i ustalić, aby wykonywał się na przykład raz w tygodniu. Pamiętajmy, aby kliknąć na **Save settings** w celu zachowania wszystkich zmian.

Scheduled clean-up settings
Take control of clean-ups: Upgrade to Premium for a more powerful and flexible scheduler
Enable scheduled clean-up and optimization (Beta feature)
Select schedule type (default is Weekly)
Weekly
Optimize database tables
Clean all post revisions
Remove auto-draft posts
Remove trashed posts
Remove spam and trashed comments
Remove unapproved comments
Remove expired transient options
Save settings

Optymalizujemy grafikę

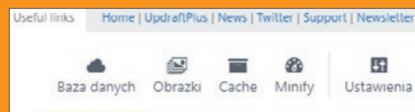
Szybko przekonamy się, że nieodpowiednia grafika to najgorsze, co może być dla czasu ładowania strony. Wystarczy, że umieścimy na jednej stronie kilka obrazków, które nie będą odpowiednio zoptymalizowane, a czas ładowania strony z niecałej sekundy może wzrosnąć nawet kilkukrotnie. Należy zawsze starać się umieszczać na swojej stronie zdjęcia w wysokiej jakości, które jednak nie będą zajmowały zbyt dużo przestrzeni dyskowej i nie będą miały zbyt dużej rozdzielczości.

1 W polu wtyczki WP-Optimize klikamy na górnym pasku na **Obrazki**.

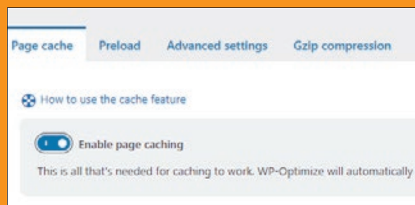


OPTYMALIZUJEMY CACHE

1 Przechodzimy do zakładki **Cache**.

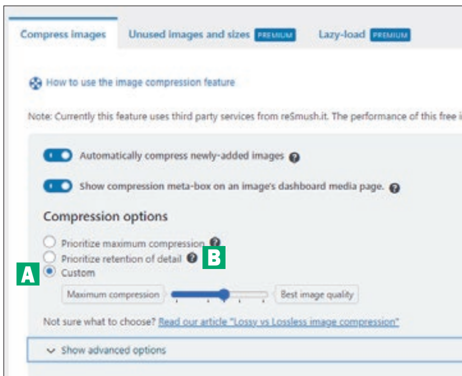


2 Jedyne, co musimy zrobić, to kliknąć na **Enable page caching**.



3 Od tej chwili wtyczka WP-Optimize będzie automatycznie dbała o zachowanie strony. Zajmuje to miejsce na dysku, ale poprawia komfort użytkowników witryny i czasy wczytywania stron przy kolejnych wizytach.

2 Następnie zaznaczamy dwie opcje dostępne w zakładce **Compress images**.

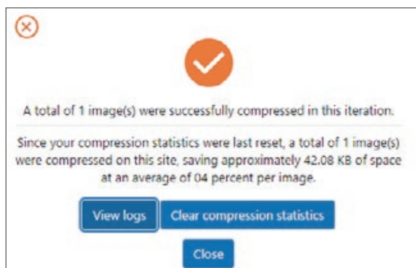


3 W polu **Compression options** wybieramy opcję **Custom** **A** i ustawiamy suwak na środku, aby zachować dobry stosunek jakości do wydajności. Jeśli zależy nam na maksymalnej wydajności, wybieramy opcję **Prioritize retention of detail** **B**.

4 Wszystkie obrazki, które już są załadowane na naszej witrynie, możemy również poddać kompresji, wystarczy je wybrać w polu **Uncompressed images**. Klikamy na **Select all**, a następnie na **Compress the selected images**.

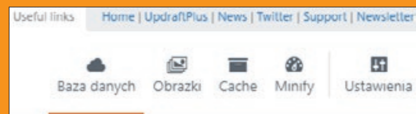


5 Po chwili zostanie wyświetlony raport na temat kompresji. Oprócz tego, że nasza strona będzie szybciej się ładowała, uzyskamy dodatkowe miejsce na pliki na dysku. Jest to ważne, jeśli wykupiliśmy ograniczoną przestrzeń serwerową i zaczyna nam brakować miejsca.

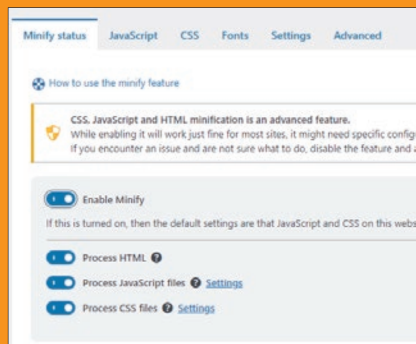


KORZYSTAMY Z MINIFY

1 Przechodzimy do zakładki **Minify**.



2 Jedyne, co musimy zrobić, to aktywować funkcję **Enable Minify**. Dzięki temu wszelkie elementy **JavaScriptu** oraz **CSS** na naszej stronie będą zoptymalizowane. To działanie przynosi świetne efekty w wypadku większości stron, które zawierają tego typu elementy. Należy jednak dokładnie zweryfikować, czy strona po aktywowaniu tej funkcji działa poprawnie. Jeśli zaobserwujemy jakieś błędy, należy wyłączyć Minify lub usunąć elementy JavaScriptu albo CSS, które mogą powodować problemy.



Kopia zapasowa i przywracanie

Jeśli chodzi o dane, nigdy nie należy być zbyt ostrożnym. Jest to trafne stwierdzenie szczególnie w przypadku stron internetowych. Nigdy nie wiemy, czy jakiś element naszej witryny nie stanie się podatny na atak, w wyniku którego zostaną usunięte wszystkie dane z naszego serwera. Dlatego też zaleca się wszystkim korzystającym z WordPressa i innych systemów CMS, które pomagają zarządzać stronami, aby często wykonywać kopie zapasowe.

W przypadku WordPressa mamy kilka możliwości wykonywania kopii zapasowych. Możemy wszystko robić ręcznie, częściowo automatycznie z poziomu panelu administratora i całkowicie automatycznie po wcześniejszej konfiguracji.

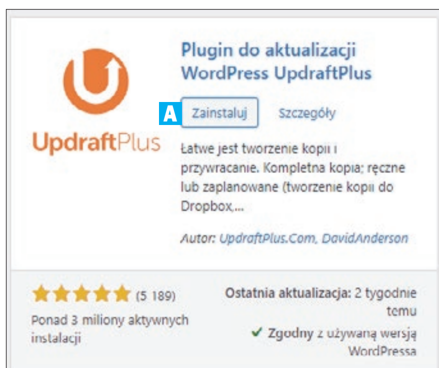
Pierwsza opcja polega na zalogowaniu się na serwer, a następnie przekopiowaniu wszystkich plików, jakie się na nim znajdują, do wybranej lokalizacji. W celu odtworzenia kopii wystarczy później wgrać wszystkie pliki ponownie na serwer. Nie jest to najwygodniejsze rozwiązanie, dlatego też zalecamy skorzystać z dostępnych wtyczek, które znacznie uproszczą cały proces.

Nie należy również przesadzać z regularnością wykonywania kopii zapasowych – jeśli do naszej strony dodajemy nowe posty średnio raz na tydzień czy dwa, tak samo często powinniśmy wykonywać kopie zapasowe.

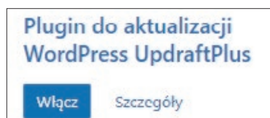
Wykonujemy kopię za pomocą wtyczki

Warto skorzystać z wtyczki **UpdraftPlus**, która jest darmowa i ma ponad 3 miliony aktywnych instalacji. Pozwala na błyskawiczne przywracanie kopii. Backup możemy wykonywać do Dropboxa, Google Drive'a, OneDrive'a, serwerów FTP, a nawet na e-mail.

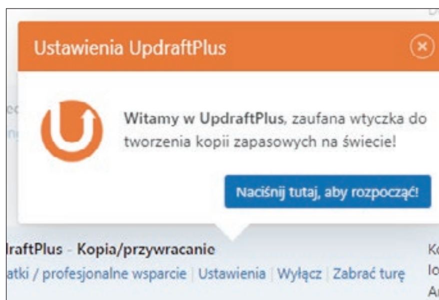
1 Logujemy się do panelu zarządzania naszej witryny, przechodzimy do zakładki wtyczki, klikamy na **Dodaj nową**, wyszukujemy wtyczkę **UpdraftPlus** i klikamy na **Zainstaluj** **A**.



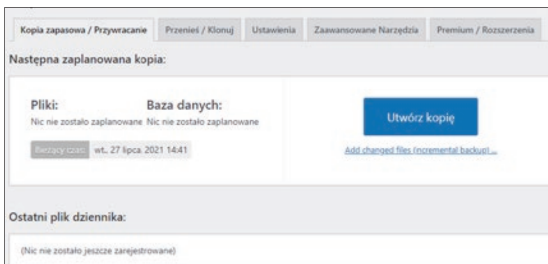
2 Następnie klikamy na **Włącz**.

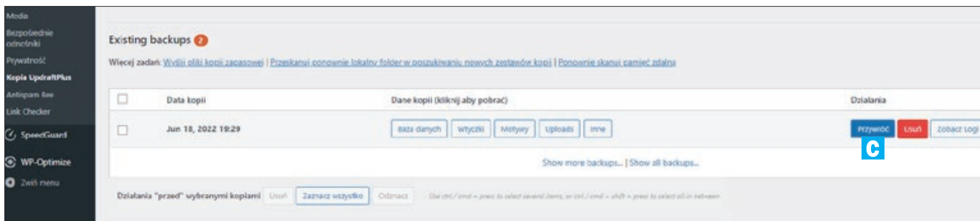


3 Teraz klikamy na **Naciśnij tutaj, aby rozpocząć!**.



4 W oknie ustawień wtyczki w zakładce **Kopia zapasowa / Przywracanie** klikamy na **Utwórz kopię**.





5 Teraz zaznaczamy opcję uwzględniania bazy danych oraz wszystkich plików i klikamy na **Utwórz kopię** **B**.

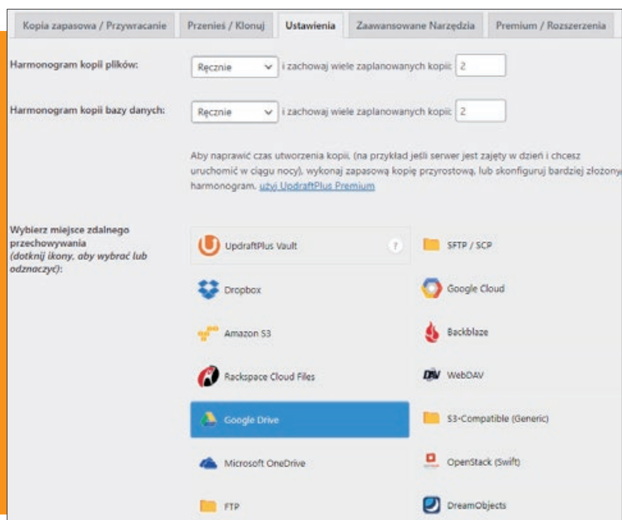


6 W celu przywrócenia kopii wystarczy przewinąć widok niżej i w polu **Existing backups** wybrać zakres przywracania, a następnie kliknąć na **Przywróć** **C**.

Warto pamiętać, że WordPress pozwala nawet mniej doświadczonym użytkownikom na tworzenie skomplikowanych witryn internetowych oraz aplikacji webowych. Wszystko dzięki ogromnej ilości gotowych do wykorzystania wtyczek i motywów. A w dalszej części książki poznamy od podstaw JavaScript i zobaczymy, jak za pomocą tego języka tworzyć interaktywne obiekty na stronach internetowych.

KORZYSTAMY Z ZAPISU W CHMURZE

W celu skorzystania z chmury przechodzimy do zakładki **Ustawienia**, następnie wybieramy dysk internetowy. Po zapisaniu zmian musimy wejść na tę zakładkę ponownie i dokonać autoryzacji na naszym dysku internetowym, aby połączyć konto i umożliwić zapis na naszym dysku. Później kopie będą trafiały automatycznie na wskazany przez nas dysk.



4 JavaScript: jak zacząć

Do tej pory poznaliśmy podstawowe technologie internetowe przydatne przy tworzeniu witryn, skorzystaliśmy również z drogi na skróty i skonfigurowaliśmy blog w serwisie Blogger. Poznaliśmy także CMS – WordPress. W tym i kolejnych rozdziałach skupimy się już na wykorzystaniu JavaScript do tworzenia stron i dynamicznych, interaktywnych obiektów na stronach internetowych

Na początku tej książki dowiedzieliśmy się bardzo ogólnie, czym jest JavaScript. Obecnie JS jest znacznie potężniejszy w porównaniu z wersjami używanymi 10 lat temu. JavaScript to bezpieczny język – działa w przeglądarce, w bezpiecznym trybie piaskownicy. Oznacza to, że ma dostęp tylko do przeglądarki i nie ma dostępu na przykład do lokalnego systemu plików.

Co pozwala wykonać JavaScript w przeglądarce?

JS w przeglądarce pozwala wykonywać bardzo wiele czynności. Umożliwia między innymi:

- interakcje z użytkownikiem
- manipulowanie stroną internetową i serwerem WWW
- dodawanie nowej zawartości HTML do strony
- reagowanie na działania użytkownika
- pobieranie i ustawianie plików cookie
- zapamiętywanie danych po stronie klienta

PORA NA JAVASCRIPT

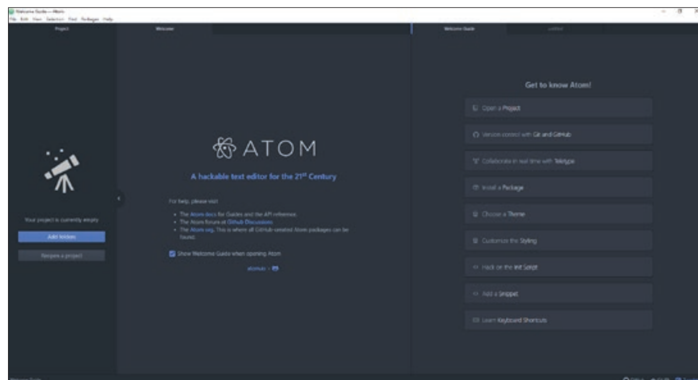
Dalsza część książki zawiera nieco bardziej zaawansowane treści i jest przeznaczona dla osób, które chcą nauczyć się więcej o JavaScripcie. Pomocna będzie podstawowa wiedza o HTML-u oraz CSS.

Czego nie może zrobić JavaScript w przeglądarce?

Jeśli JS nie może czegoś wykonać w przeglądarce, najczęściej ma to związek z bezpieczeństwem użytkownika:

- JS w przeglądarce nie może odczytywać i zapisywać dowolnych plików na dysku użytkownika, kopiować ani uruchamiać programów. Zasadniczo ma również jedynie bardzo ograniczony dostęp do funkcji systemu operacyjnego.
- Nie może chronić źródła strony ani obrazów.
- Nie może uzyskać dostępu do stron internetowych hostowanych w innej domenie.

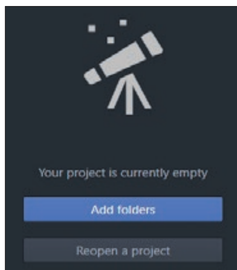
Przygotowujemy środowisko do programowania w JavaScripcie



Specjalistyczne środowisko programowania w samym JavaScripcie jest niewystarczające. Pracując nad tworzeniem rozbudowanych projektów, znacznie lepiej jest umieścić nasz projekt w IDE (zintegrowanym środowisku programistycznym), które ma wygodny w obsłudze interfejs graficzny i zapewnia wiele przydatnych funkcji, jak kolorowanie składni, podpowiadanie, debugowanie, autokorektę i wiele innych. Dodatkowo przy tworzeniu stron przydatne jest środowisko, które obsłuży HTML, CSS, JS i na przykład PHP. Wszystkie te cechy spełnia **Atom** (**DVD-KOD: 001/002**), który znajdziemy na płycie dołączonej do książki i w którym będziemy wykonywać kolejne wskazówki. Atom to nowoczesny edytor kodu źródłowego rozwijany przez popularny serwis GitHub. Oferuje duże możliwości personalizacji, kolorowanie składni wielu języków programowania, inteligentne autouzupełnianie kodu, a także otwieranie w kartach wielu plików z kodem źródłowym. Obsługuje rozszerzenia zwiększające jego możliwości.

Pierwsze kroki w Atomie – przygotowanie do pracy

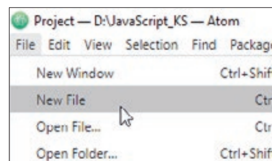
1 Po pierwszym uruchomieniu edytora Atom klikamy na **Add folders** po lewej stronie.



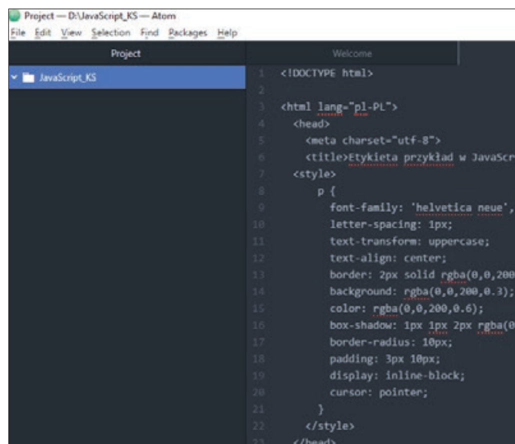
2 Wskazujemy folder i klikamy na **Wybierz folder**.



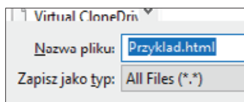
3 Teraz na górnym pasku klikamy na **File**, **New File**.



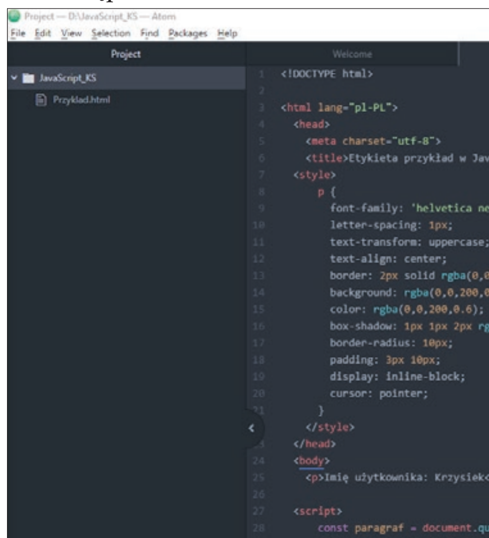
4 Właściwie już możemy zacząć pisać nasz kod, jednak nie będziemy mogli korzystać ze specjalnych narzędzi dostępnych dla konkretnego języka. Aby to było możliwe, musimy najpierw zapisać nasz plik.



W przypadku pliku HTML wystarczy zapisać go z takim właśnie rozszerzeniem.



5 Składnia naszego pliku zostanie automatycznie rozpoznana i podświetlona, a sam plik zostanie podpięty do projektu i będziemy mogli szybko uzyskać do niego dostęp.

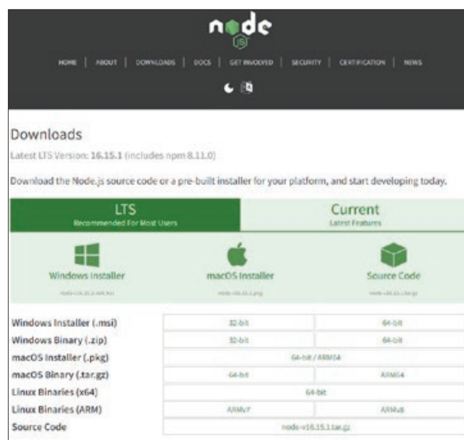


6 Po wprowadzeniu zmian i ich zapisaniu możemy w Eksploratorze otwierać daną stronę. Atom potrafi jednak znacznie więcej.

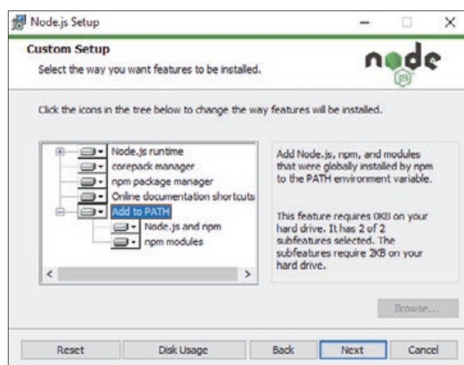
Tworzenie skryptów JS w Atomie – lepsza integracja

Do tej pory tworzyliśmy skrypty JavaScript w pliku HTML i obserwowaliśmy efekty działania takiego skryptu bezpośrednio w przeglądarce. Atom pozwala znacznie przyspieszyć prace nad skryptami, jeśli tworzymy je i testujemy bezpośrednio w tym edytorze. Co najważniejsze, umożliwia nam to wygodne debugowanie kodu i szybkie wprowadzanie poprawek. W celu pełnej integracji najprościej zainstalować moduł **Node.js** w naszym systemie, dzięki czemu globalnie uzyskamy dostęp do komend i funkcjonalności JS, oraz dodatkowy moduł **Script** w samym Atomie, aby móc szybko uruchamiać napisany kod.

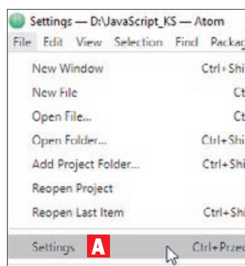
1 Zaczynamy od zainstalowania pakietu **Node.js**. Pobieramy aktualną wersję ze strony: **nodejs.org/en/download** i instalujemy.

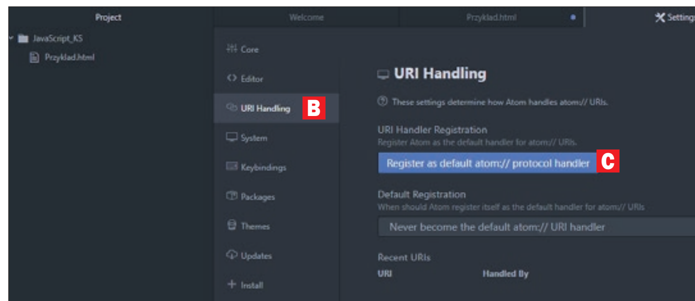


2 Najważniejsze to upewnić się podczas instalacji, że opcja **Add to PATH** jest aktywna i dopiero wtedy kliknąć na **Next**. Bez tego nie będziemy mogli bez problemu uzyskać dostępu do bibliotek JS.



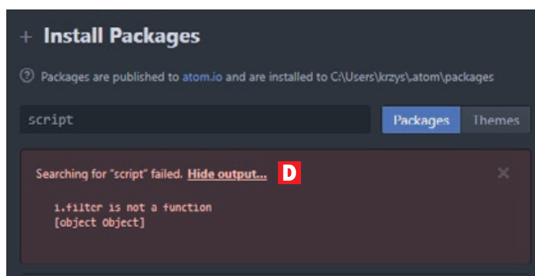
3 Następnie przechodzimy do programu Atom i klikamy na górnym pasku na **File**, **Settings** **A**.



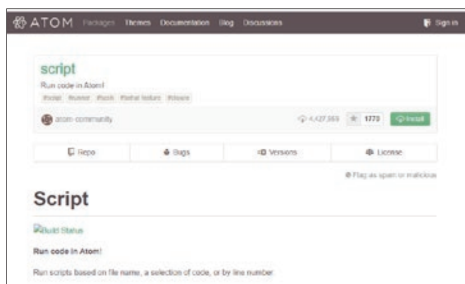


4 Teraz przechodzimy do zakładki **URI Handling** **B** i klikamy na **Register as default** **C**. **Uwaga!** Bez tego nie będziemy mogli uzyskać dostępu do pełnego katalogu dodatków.

5 Teraz w oknie ustawień klikamy na **Install**, w pole wyszukiwania wpisujemy **script** i instalujemy odpowiedni pakiet. Może jednak pojawić się nam taki błąd **D**.



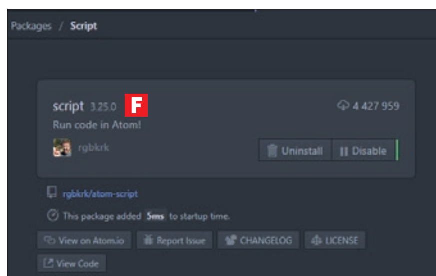
6 W takim wypadku wchodzimy na stronę **atom.io/packages/script** i klikamy na **Install**.



7 W oknie, które się pojawi, klikamy na **Otwórz Atom** **E**.



8 Jeśli wszystko przebiegnie pomyślnie, będziemy mieli od razu aktywny dodatek **script** **F**, który umożliwia uruchamianie skryptów JS bezpośrednio w programie Atom.



Integracja plików JS z HTML-em

W trakcie tworzenia kodu JS w większości przypadków nie będziemy chcieli zmieniać kodu HTML i CSS, a dodatkowo umieszczanie skomplikowanego kodu JS wewnątrz pliku HTML może stać się kłopotliwe i cały dokument będzie mniej czytelny. Dlatego też najlepszym rozwiązaniem, które pozwala zachować porządek, jest tworzenie kodu JavaScript w odrębnym pliku wewnątrz tego samego katalogu, trzeba tylko umieścić odpowiedni odnośnik do tego pliku w kodzie HTML. Zobaczmy na przykładzie naszej interaktywnej etykiety, jak to zrobić.

1 Tworzymy wewnątrz tego samego projektu nowy plik. Klikamy w Atomie na

górnym pasku na **File**, **New File**.

Project — D:\JavaScript_KS
File Edit View Selection
New Window
New File
Open File...

2 Teraz kopiujemy całą zawartość pomiędzy znacznikami **<script>** z pliku HTML i wklejamy do nowo utworzonego dokumentu. Klikamy na górnym pasku na **File**, **Save As** **A**.

3 Nadajemy nazwę dla nowego pliku z rozszerzeniem **.js** i klikamy na **Zapisz**.

Nazwa pliku: moje_skrypty.js
Zapisz jako typ: All Files (*.*)

4 Składnia pliku zostanie podświetlona i jako język programowania zostanie wskazany JavaScript.

Welcome | Przyklad.html | moje_skrypty.js

```
const paragraf = document.querySelector('p');

paragraf.addEventListener('click', zmienNazwe);

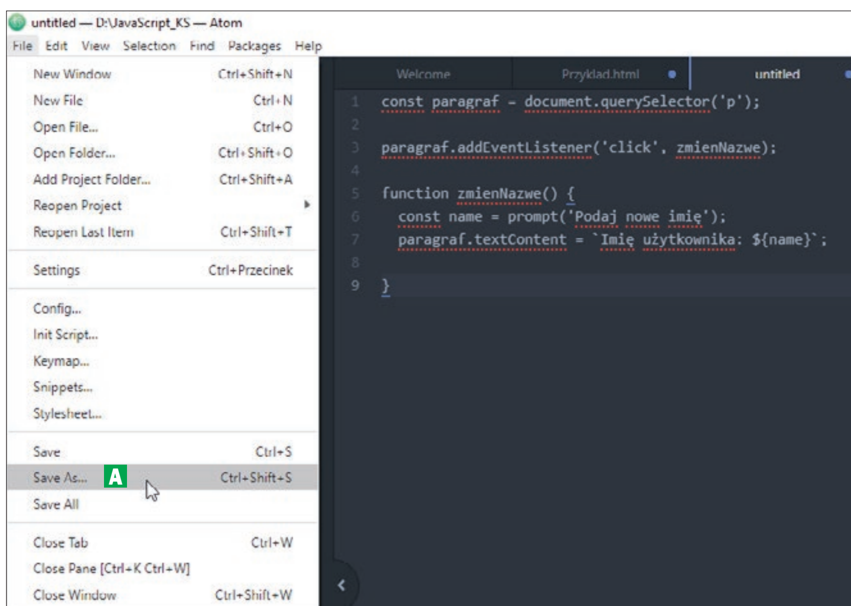
function zmienNazwe() {
  const name = prompt('Podaj nowe imię');
  paragraf.textContent = `Imię użytkownika: ${name}`;
}
```

GDZIE UMIESZCZAĆ ZNACZNIK <SCRIPT>

Praktycznie zawsze, gdy tworzymy kod JS wewnątrz dokumentu HTML, korzystamy ze znacznika <script>, co widzieliśmy w pierwszym rozdziale. Wiemy też, z czego składa się dokument HTML. W naszym przykładzie znacznik <script> umieściliśmy w sekcji „body", warto jednak już teraz wiedzieć, że równie dobrze można umieścić ten znacznik w sekcji „head". Oczywiście jedno i drugie rozwiązanie ma swoje zastosowania, które poznamy w dalszej części książki.

5 Teraz wracamy do dokumentu HTML i pomiędzy znaczniki **<script>** wstawiamy odnośnik do naszego pliku ze skryptem JS **B**.

6 Dodatkowo wskazaliśmy jako atrybut **type javascript**, gdyż wewnątrz tego znacznika mogą znajdować się również inne typy skryptów. Poprzez



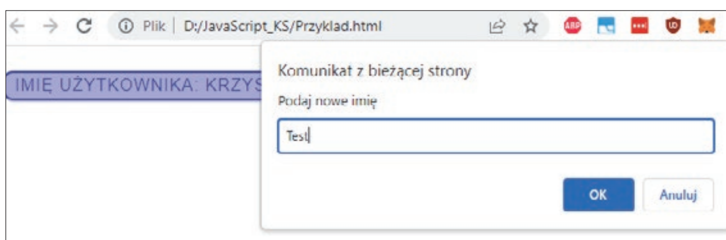
bezpośrednią deklarację podpowiadamy przeglądarce, jakiego typu języka ma się spodziewać.

7 Po zapisaniu obydwu dokumentów i uruchomienia pliku **Przyklad.html** nasza interaktywna etykieta będzie nadal działać, a w kodzie strony nie będzie widać naszego skryptu i będzie on znacznie odchudzony.

```
        }
      </style>
    </head>
    <body>
      <p>Imię użytkownika: Krzysiek</p>

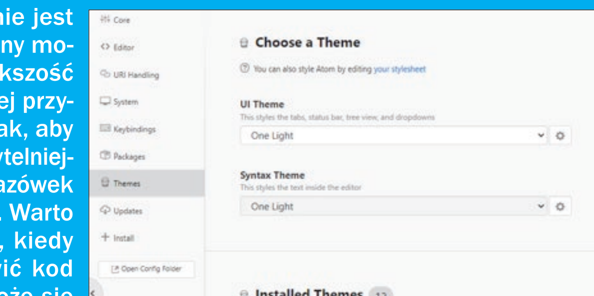
B <script type="text/javascript" src="moje_skrypty.js"></script>

    </body>
  </html>
```



ZMIANA MOTYWU GRAFICZNEGO

Domyślny motyw w Atomie jest ciemny i jest to preferowany motyw stosowany przez większość programistów jako bardziej przyjazny dla oczu. Teraz jednak, aby ilustracje w książce były czytelniejsze, na potrzeby tych wskazówek zmienimy motyw na jasny. Warto to rozważyć także wtedy, kiedy potrzebujemy przedstawić kod podczas prezentacji – może się okazać, że kod na białym tle będzie łatwiejszy w odbiorze dla nieprogramistów.



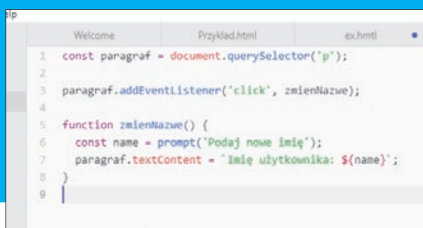
3 Po chwili nasz kod będzie wyświetlany na białym tle.

1 Klikamy na górnym pasku na **File**, **Settings**.

2 Następnie przechodzimy do zakładki **Themes** i po prawej w polu **UI Theme** wybieramy **One Dark**, a w polu **Syntax Theme** – **One Light**.

Settings — D:\JavaScri
File Edit View Selecti
New Window
New File
Open File...
Open Folder...
Add Project Folder...
Reopen Project
Reopen Last Item
Settings

4 Możemy również poeksperymentować i wybrać jeden z wielu innych dostępnych motywów lub pobrać własny.




```
const paragraf = document.querySelector('p');

paragraf.addEventListener('click', zmienNazwe);

function zmienNazwe() {
  const name = prompt('Podaj nowe imię');
  paragraf.textContent = `Imię użytkownika: ${name}`;
}
```

Wyświetlamy informacje za pomocą JavaScriptu

Wiemy już, jak w dokumencie HTML bezpośrednio wywoływać zewnętrzne pliki JS. Nie zawsze jednak konieczne jest otwieranie strony w celu przetestowania działania skryptu. W dużej mierze skrypty JS działają w tle i użytkownik nawet nie musi sobie zdawać sprawy z tego, że są one aktywne i na przykład czekają na interakcje.

W celu weryfikacji działania skryptu lub wywoływania jego poszczególnych funkcji najczęściej wykorzystuje się konsolę w przeglądarce lub w IDE. JS ma odpowiednie metody, które umożliwiają wyświetlanie informacji w konsoli, dzięki czemu programista może szybko zweryfikować poprawność logiki swojego kodu lub na konkretnym etapie działania pętli zweryfikować wartości jakichś zmiennych.

Oto kilka metod wyświetlania informacji za pomocą JS; skorzystamy z nich, żeby wyświetlić znane wszystkim **Hello World** na ekranie naszego komputera:

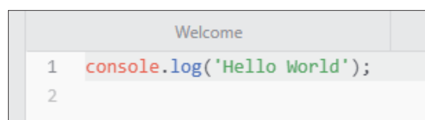
- **console.log()**
- **document.write()**
- **alert()**

Każda z powyższych metod daje możliwość wyświetlenia treści. Warto jednak wiedzieć, że **document.write()** używamy, gdy treść ma się pojawić w dokumencie HTML, natomiast **console.log()** umożliwia podgląd danych w samej konsoli bez ingerencji w wygląd dokumentu i najczęściej używa się właśnie tej metody jako narzędzia do debugowania kodu.

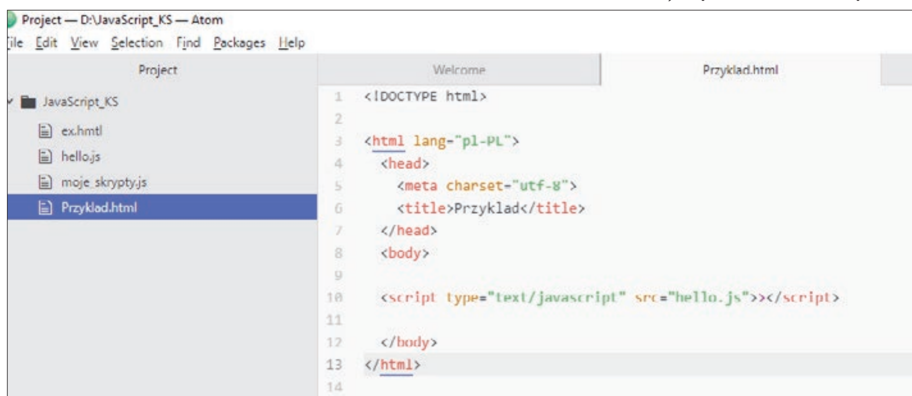
Uwaga! Wszelkie porady dotyczące przeglądarek będą opisywane na przykładzie Google Chrome – w innych przeglądarkach, jak Firefox, Opera, Edge, opisywane kroki również są możliwe do realizacji, jednak mogą się nieznacznie różnić.

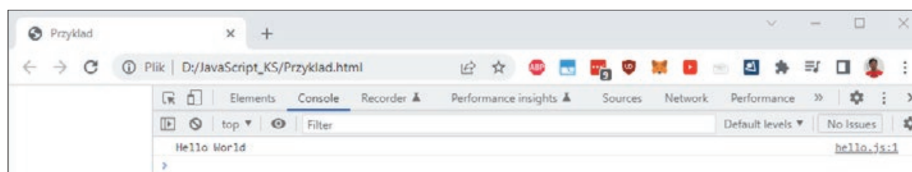
Console.log()

1 W oknie z plikiem JS wprowadzamy treść – **console.log(‚Hello World’);** Zapisujemy skrypt i uruchamiamy stronę w przeglądarce.

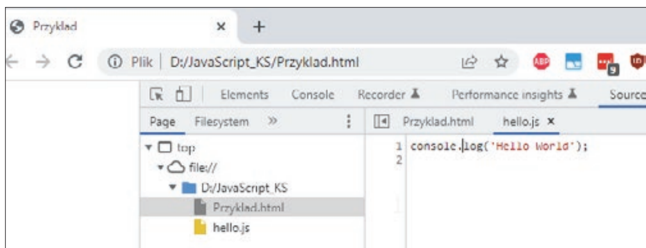


2 Ponieważ niczego nie podaliśmy w treści dokumentu, a jedynie zawarliśmy nasz

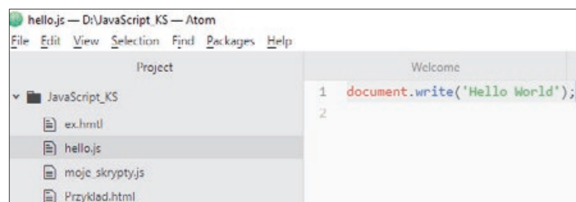




plik JS, strona powinna na pierwszy rzut oka być pusta. W Chrome wciskamy kombinację klawiszy ctrl+shift+J. Pojawi się widok z otwartą konsolą – **Console** – oraz zakodowana przez nas informacja.



3 Klikając po prawej stronie na odnośnik do pliku **JS**, zobaczymy również linię kodu, która jest odpowiedzialna za wyświetlenie tej informacji. Jest to bardzo wygodna forma weryfikowania różnego typu relacji podczas działania skryptu JS.



Document.write()

1 W oknie z plikiem JS wprowadzamy treść **document.write('Hello World');**

2 Zapisujemy i otwieramy dokument HTML – zostanie wy-

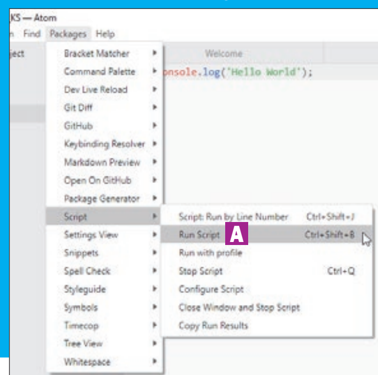
URUCHAMIAMY SKRYPT JS BEZPOŚREDNIO W ATOMIE

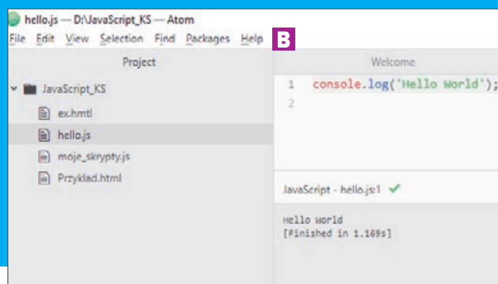
Pokazane powyżej najprostsze przykłady wywoływania kodu JavaScript i wyświetlania informacji bezpośrednio z kodu zostały zaprezentowane na stronie internetowej wewnątrz przeglądarki.
Jest jednak szybszy sposób weryfikacji poprawności kodu. Działanie samego skryptu JS możemy sprawdzać bezpośrednio w Atomie.

1 Jeśli zgodnie z wcześniejszą poradą zainstalowaliśmy dodatek **script**, będziemy mogli wykonywać kod JS bezpośrednio w programie Atom.

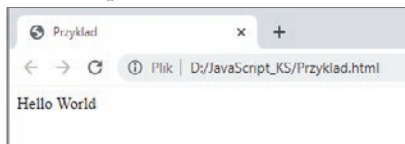
2 Po wpisaniu kodu i zapisaniu go w dokumencie z rozszerzeniem JS wystarczy kliknąć na górnym pasku na **Packages**, **Script**, **Run Script** **A**.

3 Jeśli wykonaliśmy konfigurację poprawnie, powinniśmy otrzymać taki rezultat **B**.
Dzięki Node.js jesteśmy w stanie bezpośrednio w Atomie uzyskać rezultat, jaki moglibyśmy zaobserwować w konsoli przeglądarki.





świetlony wprowadzony przez nas tekst. Jak widać, ta metoda zapisuje wprowadzony ciąg znaków bezpośrednio na stronie.

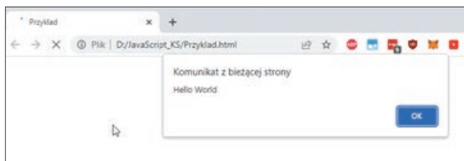


Alert()

1 W oknie z plikiem JS wprowadzamy treść **document.write('Hello World');**



2 Zapisujemy i otwieramy dokument HTML – zostanie wyświetlony w formie okna pop-up. Jak widać, ta metoda tworzy okno z komunikatem, który należy zatwierdzić, aby strona mogła dalej się ładować.



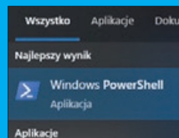
Dla naszych potrzeb najczęściej będziemy korzystać z opcji **console.log()**, która pozwala na szybki podgląd bez konieczności modyfikowania widoku strony i zatwierdzania jakichkolwiek informacji.

W PRZYPADKU BŁĘDU WYKONANIA

Może się zdarzyć, że pomimo wykonania opisanych kroków otrzymamy błąd **A**.

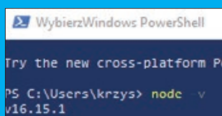
```
'node"' is not recognized as an internal or external command,
operable program or batch file. A
[Finished in 0.075s]
```

Należy wtedy zweryfikować, czy w naszym systemie jest poprawnie zainstalowany pakiet Node.js oraz skonfigurowana ścieżka dostępu w PATH.

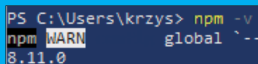


1 Uruchamiamy **Windows PowerShell**.

2 Następnie wpisujemy polecenie **node -v** i zatwierdzamy klawiszem enter.

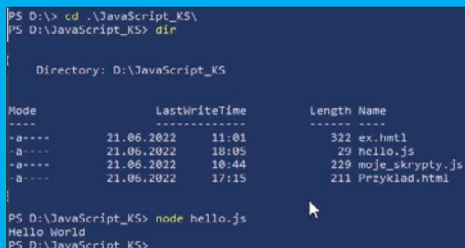


3 Powinniśmy otrzymać informację o zainstalowanej wersji Node.js, następnie zatwierdzamy polecenie **npm -v**.



4 Tutaj też powinniśmy otrzymać informacje o wersji modułu **npm**. Jeśli otrzymamy te komunikaty, przechodzimy do lokalizacji naszego przykładowego pliku JS (korzystamy w tym celu z komendy **cd**) i uruchamiamy nasz skrypt poleceniem **node hello.js**.

5 Jak widać, skrypt został wykonany. Jeśli więc w samym Atomie uzyskujemy błąd, należy ponownie uruchomić komputer i jeżeli to nie pomoże, ponownie zainstalować program Atom. Jeżeli natomiast po wpisywaniu komend sprawdzających wersję **node** oraz **npm** nie otrzymamy informacji o wersji, należy zweryfikować poprawność instalacji **Node.js**.



Warto wiedzieć – konsola w przeglądarce

Poznaliśmy już konsolę w przeglądarce, która umożliwia odczytywanie informacji zawartych między innymi w skryptach JavaScript. Jest to ważne narzędzie, które zaawansowanym użytkownikom pozwala na bardzo wiele. Pozwala na przykład manipulować danymi czy wywoływać funkcje.
Zanim przejdziemy do szczegółów, warto wiedzieć, że możemy sami wykonywać funkcje i metody JavaScript bezpośrednio w konsoli.
Dodatkowo to, że obecnie dokumenty HTML mogą być opisywane jako modele obiektowe dzięki standardowi DOM, sprawia, że możemy odwoływać się do każdego elementu jak do obiektu przez parametry. Każdy element na stronie staje się dostępny do naszej dyspozycji jako osobny obiekt. Więcej o tym standardzie i o tym, co można zrealizować w jego ramach w JS, przeczytamy na dalszych stronach. Teraz jedynie warto zapamiętać, że ten standard pozwala nam naprawdę na wiele podczas korzystania z konsoli.

Zmieniamy tryb wyświetlania w tryb edytowania

Jeden z najbardziej spektakularnych przykładów tego, jak strony internetowe korzystają z DOM, to możliwość zmiany naszej przeglądarki i wyświetlanej strony na edytowalny dokument.

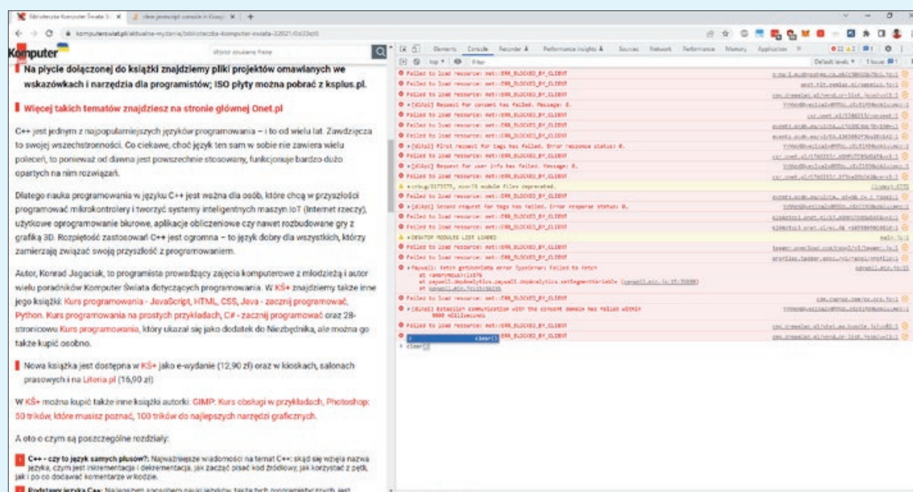
1 Otwieramy stronę internetową, na której chcemy przetestować działanie trybu edytowania. Przechodzimy do konsoli. W Chrome wystarczy wcisnąć kombinację ctrl+shift+J.

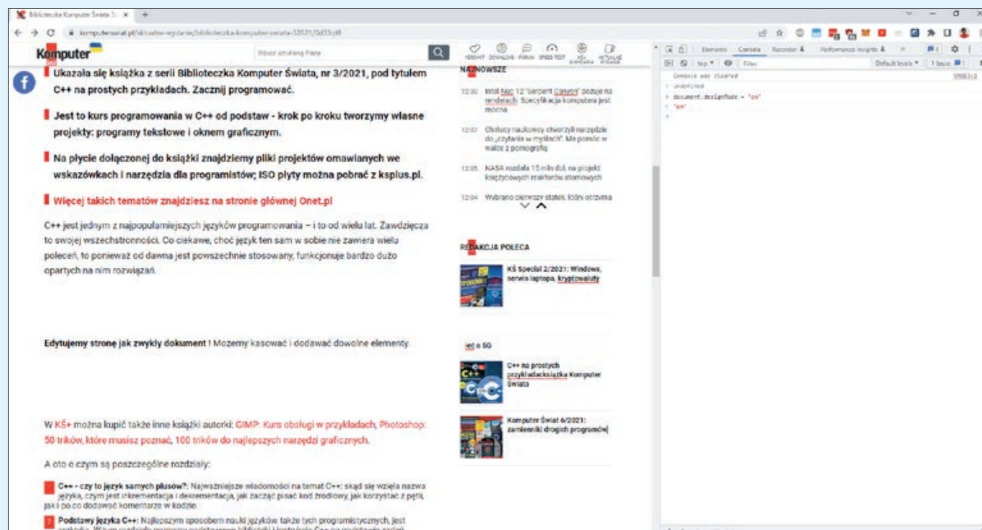
2 Jeśli mamy dużo różnych dodatków w przeglądarce, nasza konsola może nie być przejrzysta, gdy zwrócą one zbyt wiele informacji. Możemy wyczyścić widok. Wystarczy wpisać komendę **clear()** i zatwierdzić.

3 Po jej wykonaniu konsola będzie wyczyszczona. Teraz w celu aktywowania trybu edycji zatwierdzamy polecenie **document.designMode = 'on'**

```
> document.designMode = 'on'
<· 'on'
>
```

4 Od tej pory możemy dowolnie manipulować elementami na stronie. Ponieważ zgodnie ze standardem DOM wszystko w dokumencie HTML jest obiektem, może-



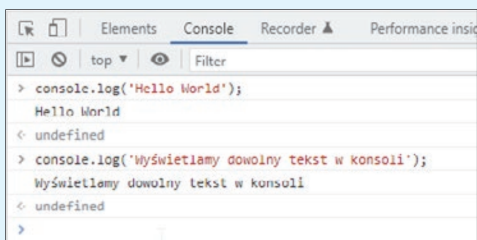


my każdy obiekt usunąć lub edytować jego zawartość.

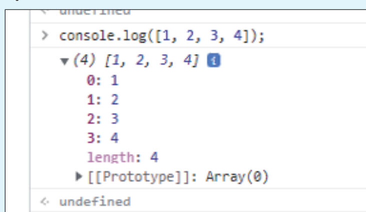
Wykonujemy polecenia JavaScriptu w konsoli

Korzystając z konsoli, możemy przeprowadzić wiele prostych i złożonych zadań.

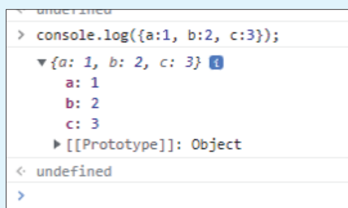
1 Możemy bezpośrednio wykonywać polecenia, które wcześniej wprowadzaliśmy w pliku JS, jak **console.log('Hello World');**



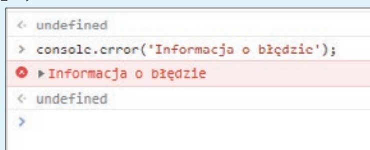
2 Możemy w ten sposób tworzyć i odczytywać tablice.



3 Możliwe jest też tworzenie obiektów.



4 Warto wiedzieć, że możemy zmodyfikować metodę i korzystać na przykład z opcji **error**.



5 To może być szczególnie przydatne, jeśli tworzymy skomplikowany skrypt.

6 Warto również wiedzieć, że mamy możliwość tworzenia w dość prosty sposób tabel **A**.

7 Możemy również liczyć wywołania pętli **B**.

```
< undefined
> console.table({'a':1, 'b':2});
```

A

(index)	Value
a	1
b	2

```
▸ Object
< undefined
>
```

```
< undefined
> for(let i=0;i<5;i++){
      console.count(i);
  }
  0: 1
  1: 1
  2: 1
  3: 1
  4: 1
< undefined
> |
```

B

8 Możemy też deklarować funkcje, a później je wywoływać.

```
> function loop(){
      var x = 5
      for(let i=1;i<6;i++){
          x = x * i;
          console.log(x);
      }
  }
< undefined
> loop()
  5
  10
  30
  120
  600
< undefined
>
```

Analizujemy czas wykonywania konkretnej funkcji

Podczas tworzenia skryptów JavaScript może się okazać, że wydajność i optymalizacja naszego skryptu mają duże znaczenie. Wystarczy umieszczenie kilku mało wydajnych funkcji, a czas wykonywania konkretnej akcji na stronie się wydłuży, co może zniechęcić użytkowników naszej witryny.
W JavaScripcie jest wiele metod, które umożliwiają weryfikację czasu wykonywania poszczególnych elementów kodu. Jeśli chcemy zmierzyć czas wykonywania poszczególnej funkcji, najprościej skorzystać z możliwości metody **console.time()**

1 W konsoli wprowadzamy polecenie **console.time('czasPetli')**, następnie w kolejnym wierszu, do którego przechodzimy, naciskając klawisze shift+enter, wpisujemy wywołanie funkcji, której czas chcemy zmierzyć, a następnie w ostatniej linii – polecenie **console.timeEnd('czasPetli')**, co pozwala na zmierzenie czasu realizowania danej funkcji. Wartość **czasPetli** jest dowolna i ustalana przez użytkownika – musi jednak zgodzić się w obydwu poleceniach.

```
> console.time('czasPetli')
  loop()
  console.timeEnd('czasPetli')
  5
  10
  30
  120
  600
  czasPetli: 0.14501953125 ms
< undefined
> |
```

2 Czas działania dla 80 iteracji rośnie i możemy to zaobserwować, nie jest to jednak duży wzrost, gdyż nie mamy tutaj do czynienia ze wzrostem wykładniczym, jaki mógłby wystąpić przy pętli zagnieżdżonej.

```
  9.427473508330258e+111
  7.259154601414298e+113
  5.6621405891031526e+115
  4.4730910653914907e+117
  czasPetli: 1.07275390625 ms
< undefined
>
```

5 JavaScript: podstawy

Znając już obsługę środowiska IDE Atom oraz zasady korzystania z konsoli, możemy przejść do podstaw JavaScriptu. Poznamy krok po kroku najważniejsze elementy tego języka, które przydadzą nam się przy tworzeniu własnych skryptów

Zmienne w JavaScripcie

Zmienne w JavaScripcie to kontenery, które przechowują dane wielokrotnego użytku.

- Wartość przechowywaną w zmiennej można zmienić podczas wykonywania programu.
- Zmienna to tylko nazwa nadana lokalizacji w pamięci, wszystkie operacje wykonane na zmiennej wpływają na tę lokalizację w pamięci.

```
Welcome                                   Przyklad.html
console.log('Zmienne w przykładach');
x = 2

JavaScript - hello.js:3

Zmienne w przykładach
[stdin]:4
x = 2;
 ^

ReferenceError: x is not defined  A
    at [stdin]:4:3
    at Script.runInThisContext (node:vm:129:12)
    at Object.runInThisContext (node:vm:305:38)
    at node:internal/process/execution:76:19
    at [stdin]-wrapper:6:22
    at evalScript (node:internal/process/execution:75:60)
    at node:internal/main/eval_stdin:29:5
    at Socket.<anonymous> (node:internal/process/execution:213:5)
    at Socket.emit (node:events:539:35)
    at endReadableNT (node:internal/streams/readable:1345:12)
[Finished in 1.229s]
```

- W JavaScripcie wszystkie zmienne muszą być zadeklarowane przed użyciem.

Najważniejsza z tych własności to ta ostatnia – bez zadeklarowania zmiennej nie możemy jej użyć i pojawi się błąd. W tym przypadku **x is not defined** **A**, czyli x nie został wcześniej zadeklarowany.

Deklaracja poprzez słowo kluczowe var

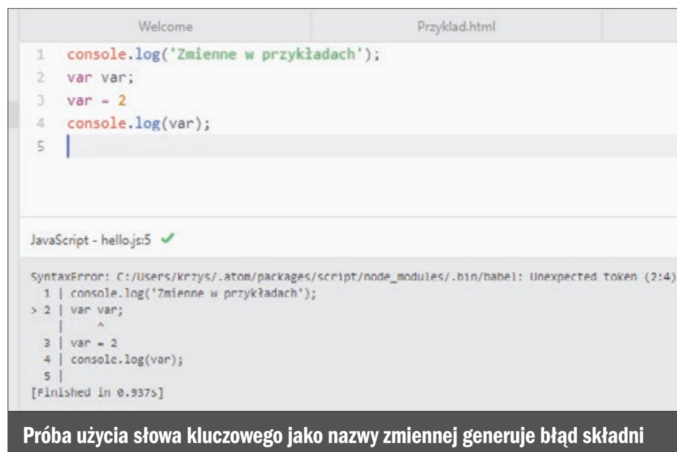
Przed wejściem w życie standardu **ES2015** (jest to usystematyzowana wersja standaryzacji dla JavaScriptu) mogliśmy deklarować zmienne w JS jedynie za pomocą słowa kluczowego **var**, po którym następuje nazwa zmiennej

```
Welcome
console.log('Zmienne w przykładach');
var x;
x = 2
console.log(x);

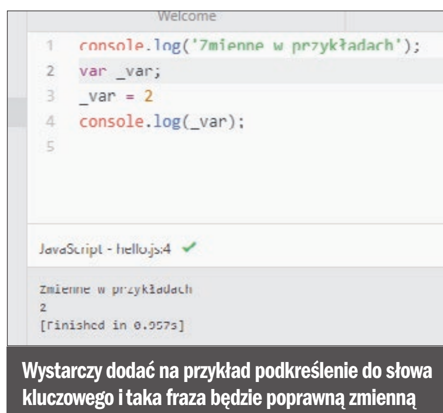
JavaScript - hello.js:4

Zmienne w przykładach
2
[Finished in 0.948s]
```

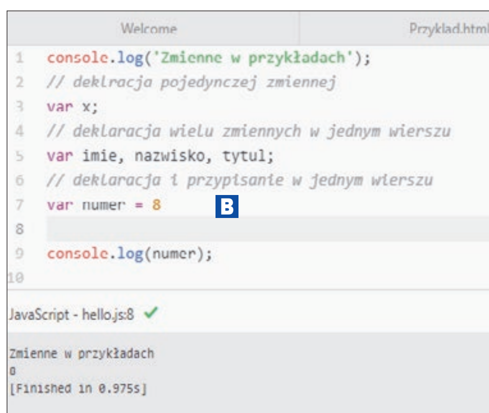
i średnik. Obecnie możemy również korzystać ze słów **let** oraz **const**.
W tym przykładzie **x** to nazwa zmiennej, która powinna być zdefiniowana przez użytkownika i powinna być unikalna. Tego typu nazwy są również znane jako identyfikatory. Zasady tworzenia identyfikatora w JavaScripcie są takie, że nazwa identyfikatora nie powinna być żadnym predefiniowanym słowem (tak zwanym słowem kluczowym).



Próba użycia słowa kluczowego jako nazwy zmiennej generuje błąd składni



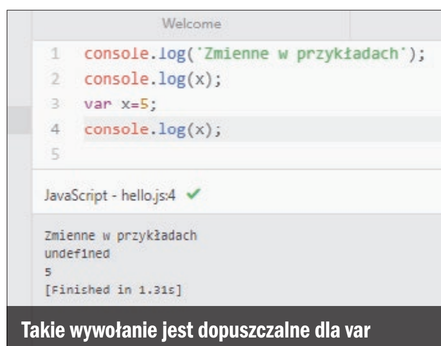
Wystarczy dodać na przykład podkreślenie do słowa kluczowego i taka fraza będzie poprawną zmienną



Pierwszym znakiem musi być litera, podkreślenie (**_**) lub znak dolara (**$**). Kolejne znaki mogą być dowolną literą lub cyfrą, podkreśleniem lub znakiem dolara.
Dodatkowo warto wiedzieć, że w JavaScripcie możemy przypisywać wartość do zmiennej zarówno w trakcie jej deklaracji, jak też później, w dowolnym miejscu skryptu **B**. Oto kilka przykładów poprawnej deklaracji zmiennych oraz przypisywania im wartości.

Deklaracja poprzez słowo kluczowe let

Na tym etapie nie będziemy dokładnie omawiać deklarowania zmiennych w ten sposób. Najważniejszą informacją powinno być dla



Takie wywołanie jest dopuszczalne dla var

nas to, że w odróżnieniu od var, które pozwala na tworzenie zmiennej o zasięgu funkcji, **let** tworzy zmienne o zasięgu bloku kodu.

JAVASCRIPT, CZYLI JĘZYK DYNAMICZNIE TYPOWANY

JavaScript jest językiem dynamicznie typowanym. Oznacza to, że nie musimy deklarować typu zmiennej, ponieważ jest on automatycznie konwertowany do pożądanych wartości w czasie wykonywania się skryptu. Przykładowo możemy zdefiniować zmienną w podany sposób, a następnie przypisać jej inną wartość, która będzie postrzegana jako wartość innego typu:

```
console.log('Zmienne w przykładach');

var zmienna = 8;
console.log(zmienna);

zmienna = 'Nowa wartosc';
console.log(zmienna);

JavaScript - hello.js:3 ✓

Zmienne w przykładach
8
Nowa wartosc
[Finished in 1.460s]
```

Jak widać, zmienna może przyjmować różne wartości i nie musimy się martwić deklarowaniem typu tak, jak ma to miejsce w innych językach programowania.
Warto też wiedzieć, co się stanie w przypadku próby dodania do siebie dwóch wartości różnych typów, na przykład **string** oraz **int**.

```
console.log('Zmienne w przykładach');
var a = 'Nowa wartosc to: ';
var b = 8
console.log(a);
console.log(b);
console.log(a+b);

JavaScript - hello.js:6 ✓

Zmienne w przykładach
Nowa wartosc to:
8
Nowa wartosc to: 8
[Finished in 0.921s]
```

Podczas operacji dodawania dwóch zmiennych nie otrzymaliśmy błędu – jest to związane ze sposobem dynamicznego zmieniania typów zmiennych. W przypadku gdy zamierzamy dodać do siebie zmienne typu **string** zawierające znaki oraz zmienne typu **int**, **float** zawierające liczby, dochodzi do konwersji tych drugich na łańcuch znaków. Dzięki temu nie pojawia się błąd, ale następuje konkatenacja (łączenie) łańcuchów.
Jeśli zależy nam na traktowaniu konkretnych wartości przechowywanych w formie łańcuchów znaków jako wartości liczbowych, możemy skorzystać z metod **parseInt()** oraz **parseFloat()**, ta pierwsza zwraca liczbę całkowitą, druga zmiennoprzecinkową.

```
console.log('Zmienne w przykładach');
var a = '1.1';
var b = '1.1'
console.log(a);
console.log(b);
console.log(a+b);
console.log("Zmiana na liczby");
console.log(parseFloat(a)+parseFloat(b));

JavaScript - hello.js:7 ✓

Zmienne w przykładach
1.1
1.1
1.11.1
Zmiana na liczby
2.2
[Finished in 0.922s]
```

Dzięki takiemu zastosowaniu metod możemy bez problemu dynamicznie korzystać praktycznie z każdej wartości przypisanej do dowolnej zmiennej.
Dodatkowo możemy skorzystać z nieco szybszej automatycznej formy konwersji. W tym przypadku dodanie operatora + przed zmienną powoduje oznaczenie jej jako wartości liczbowej.

```
console.log('Zmienne w przykładach');
var a = '1.1';
var b = '1.1'
console.log(a);
console.log(b);
console.log(a+b);
console.log("Zmiana na liczby");
console.log((+a)+(+b));

JavaScript - hello.js:8 ✓

Zmienne w przykładach
1.1
1.1
1.11.1
Zmiana na liczby
2.2
[Finished in 0.932s]
```

```
console.log('Zmienne w przykładach');
console.log(x);
let x=5;
console.log(x);
```

```
JavaScript - hello.js:5

Zmienne w przykładach
[stdin]:4
console.log(x);
            ^

ReferenceError: Cannot access 'x' before initialization
    at [stdin]:4:13
    at Script.runInThisContext (node:vm:129:12)
    at Object.runInThisContext (node:vm:305:38)
    at node:internal/process/execution:76:19
    at [stdin]-wrapper:6:22
    at evalScript (node:internal/process/execution:75:60)
    at node:internal/main/eval_stdin:29:5
    at Socket.<anonymous> (node:internal/process/execution:213:5)
    at Socket.emit (node:events:539:35)
    at endReadableNT (node:internal/streams/readable:1345:12)
[Finished in 0.939s]
```

W przypadku let otrzymamy błąd

Deklaracja poprzez słowo kluczowe const

Z tego typu deklaracji już korzystaliśmy w naszych przykładach. Wyróżnia je to, że są to zmienne tylko do odczytu – nie możemy zmieniać ich wartości w trakcie pracy skryptu.

```
console.log('Zmienne w przykładach');
const imie = 'Krzysiek'
console.log(imie);
imie = 'Adam'
console.log(imie);
```

```
JavaScript - hello.js:5

Zmienne w przykładach
Krzysiek
[stdin]:6
imie = 'Adam';
     ^

TypeError: Assignment to constant variable.
    at [stdin]:6:6
    at Script.runInThisContext (node:vm:129:12)
    at Object.runInThisContext (node:vm:305:38)
    at node:internal/process/execution:76:19
    at [stdin]-wrapper:6:22
    at evalScript (node:internal/process/execution:75:60)
    at node:internal/main/eval_stdin:29:5
    at Socket.<anonymous> (node:internal/process/execution:213:5)
    at Socket.emit (node:events:539:35)
    at endReadableNT (node:internal/streams/readable:1345:12)
[Finished in 0.95s]
```

Przy próbie przypisania nowej wartości do zmiennej const pojawia się błąd

Zakresy zmiennych w JavaScripcie

Poprzez zakres zmiennej rozumiemy taką część programu, w której uzyskamy bezpośredni dostęp do zmiennej. W JS możemy wyróżnić dwa rodzaje zakresów:

- **Zakres globalny** – zakres poza najbardziej zewnętrzną funkcją
- **Zakres lokalny** – zakres wewnątrz konkretnej funkcji

Oto praktyczny przykład:

```
console.log('Zakresy zmiennych');
let zmiennaGlobalna = "To jest zmienna globalna";

function wewn() {
  let zmiennaLokalna = "To jest zmienna lokalna";

  console.log(zmiennaGlobalna);
  console.log(zmiennaLokalna);
}

wewn();
```

```
JavaScript - hello.js:10

Zakresy zmiennych
To jest zmienna globalna
To jest zmienna lokalna
[Finished in 0.946s]
```

Deklaracja zmiennej globalnej jest poza funkcją, a lokalnej – wewnątrz funkcji. W tym przykładzie po danych wyjściowych widzimy, że zarówno jedna, jak i druga zmienna jest dostępna wewnątrz funkcji, gdyż otrzymujemy rezultat w konsoli.

```
console.log('Zakresy zmiennych');
let zmiennaGlobalna = "To jest zmienna globalna";

function wewn() {
  let zmiennaLokalna = "To jest zmienna lokalna";
}

wewn();
console.log(zmiennaGlobalna);
console.log(zmiennaLokalna);
```

```
JavaScript - hello.js:10

Zakresy zmiennych
To jest zmienna globalna
[stdin]:12
console.log(zmiennaLokalna);
            ^

ReferenceError: zmiennaLokalna is not defined
    at [stdin]:12:13
    at Script.runInThisContext (node:vm:129:12)
```

Po przeniesieniu instrukcji **console.log** zaraz po wywołaniu funkcji i wykonaniu

skryptu otrzymujemy błąd – jesteśmy w stanie zobaczyć wartość zmiennej globalnej, jednak dla zmiennej lokalnej otrzymujemy informację o tym, że nie została zadeklarowana. Tak więc, jak przeczytaliśmy powyżej, zmienna lokalna istnieje jedynie wewnątrz konkretnej funkcji, w której została zadeklarowana.

Operatory w JavaScripcie

W JavaScripcie jest mnóstwo operatorów i każdy z nich ma swoje zastosowanie. Poznamy teraz te najbardziej przydatne, zwłaszcza na początku nauki tego języka. Dzięki zapoznaniu się z operatorami już teraz, na późniejszym etapie – przy pętlach, instrukcjach warunkowych i funkcjach – będzie nam znacznie łatwiej czytać kod.
Możemy wyróżnić operatory przypisania, porównania, arytmetyczne, bitowe, logiczne, warunkowe i kilka innych.

```
Welcome                                    Przyklad.html

console.log('Operatory');
console.log('Przypisanie');
let obj = {};

obj.x = 3;
console.log(obj.x); // Wyświetli 3.
console.log(obj); // Wyświetli obiekt { x: 3 }.

const key = "y";
obj[key] = 5;
console.log(obj[key]); // Wyświetli 5.
console.log(obj); // Wyświetli { x: 3, y: 5 }.

JavaScript - hello.js:12 ✔

Operatory
Przypisanie
3
{ x: 3 }
5
{ x: 3, y: 5 }
[Finished in 1.031s]
```

Operator przypisania

To operator, z którego już korzystaliśmy; jest określany znakiem **=** i przypisuje wartość

OPERATORY PORÓWNANIA

OPERATOR	OPIS	PRZYKŁADY ZWRACAJĄCE PRAWDĘ
Równe (**==**)	Zwraca true, jeśli operandy są równe	3 == var1 „3" == var1 3 == '3'
Nierówne (**!=**)	Zwraca true, jeśli operandy nie są równe	var1 != 4 var2 != „3"
Ścisłe równe (**===**)	Zwraca true, jeśli operandy są równe i tego samego typu	3 === var1
Ścisłe nierówne (**!==**)	Zwraca true, jeśli operandy są tego samego typu, ale nie są równe lub są innego typu	var1 !== „3" 3 !== '3'
Większe niż (**>**)	Zwraca true, jeśli lewy operand jest większy niż prawy operand	var2 > var1 „12" > 2
Większe lub równe (**>=**)	Zwraca true, jeśli lewy operand jest większy lub równy prawemu operandowi	var2 >= var1 var1 >= 3
Mniejsze niż (**<**)	Zwraca true, jeśli lewy operand jest mniejszy niż prawy operand	var1 < var2 „2" < 12
Mniejsze lub równe (**<=**)	Zwraca true, jeśli lewy operand jest mniejszy lub równy prawemu operandowi	var1 <= var2 var2 <= 5

po jego prawej stronie do elementu po lewej stronie. Może pełnić wiele funkcji, jednak najczęściej będziemy go stosować w celu przypisania określonej wartości do zmiennej, obiektu lub innego elementu.

Operatory porównania

Operator porównania porównuje swoje operandy i zwraca wartość logiczną na podstawie tego, czy porównanie jest prawdziwe. Operandy mogą być wartościami liczbowymi, łańcuchowymi, logicznymi lub obiektowymi. Ciągi są porównywane na podstawie standardowej kolejności leksykograficznej, przy użyciu wartości Unicode. W większości przypadków, jeśli oba operandy nie są tego samego typu, JavaScript próbuje przekonwertować je na odpowiedni typ do porównania. To zachowanie zwykle powoduje liczbowe porównywanie operandów.

Operatory arytmetyczne

Operator arytmetyczny przyjmuje jako operandy wartości liczbowe (dosłowne lub zmienne) i zwraca pojedynczą wartość liczbową. Standardowe operatory arytmetyczne to dodawanie (+), odejmowanie (–), mnożenie (*) i dzielenie (/). Operatory te działają tak samo, jak w większości innych języków programowania, gdy są używane z liczbami zmiennoprzecinkowymi (co ciekawe, dzielenie przez zero **A** daje Infinity, czyli nieskończoność, co jest dość unikalne, bo w wielu innych językach taka operacja zwraca błąd krytyczny).

Welcome

```
console.log('Operatory');
console.log('Arytmetyczne');
var x = 4;
var y = 0; A
var z1 = y/x;
var z2 = x/y;

console.log(z1);
console.log(z2);
```

JavaScript - hello.js:9 ✓

```
Operatory
Arytmetyczne
0
Infinity
[Finished in 1.011s]
```

Oprócz standardowych operacji arytmetycznych (+, –, *, /) JavaScript udostępnia operatory arytmetyczne wymienione w tabeli.

OPERATORY ARYTMETYCZNE

OPERATOR	OPIS	PRZYKŁAD
Pozostałe (%)	Operator binarny. Zwraca całkowitą resztę z dzielenia dwóch operandów	12 % 5 zwraca 2
Przyrost (++)	Operator jednoargumentowy. Dodaje jeden do swojego operandu. Jeśli jest używany jako operator przedrostkowy (++x), zwraca wartość swojego operandu po dodaniu jednego; jeśli jest używany jako operator przyrostka (x++), zwraca wartość swojego operandu przed dodaniem jednego	Jeśli x wynosi 3, to ++x ustawia x na 4 i zwraca 4, podczas gdy x++ zwraca 3 i dopiero wtedy ustawia x na 4
Zmniejsz (--)	Operator jednoargumentowy. Odejmuje jeden od swojego operandu. Zwracana wartość jest analogiczna jak w przypadku operatora inkrementacji	Jeśli x wynosi 3, to --x ustawia x na 2 i zwraca 2, podczas gdy x-- zwraca 3 i dopiero wtedy ustawia x na 2
Negacja jednoargumentowa (-)	Operator jednoargumentowy. Zwraca negację swojego operandu	Jeśli x wynosi 3, -x zwraca -3
Jednoargumentowy plus (+)	Operator jednoargumentowy. Próbuje przekonwertować operand na liczbę, jeśli jeszcze nie jest liczbą	+"3" zwraca 3 +true zwraca 1
Operator potęgowania (**)	Oblicza podstawę do wykładnika potęg	2 ** 3 zwraca 8 10 ** -1 zwraca 0.1

OPERATORY LOGICZNE

OPERATOR	STOSOWANIE	OPIS
Logiczne ORAZ (**&&**)	expr1 && expr2	Zwraca expr1, jeśli można go przekonwertować na false; w przeciwnym razie zwraca expr2. Tak więc, gdy jest używany z wartościami boolowskimi, && zwraca true, jeśli oba operandy są prawdziwe; w przeciwnym razie zwraca false
Logiczne LUB (**||**)	expr1 | | expr2	Zwraca expr1, jeśli można go przekonwertować na true; w przeciwnym razie zwraca expr2. Tak więc, gdy jest używany z wartościami boolowskimi, | | zwraca true, jeśli któryś z operandów jest prawdziwy; jeśli oba są fałszywe, zwraca false
Logiczne NIE (**!**)	!expr	Zwraca false, jeśli jego pojedynczy operand ma wartość logiczną true; w przeciwnym razie zwraca true

Operatory logiczne

Operatory logiczne są zwykle używane z wartościami boolowskimi (logicznymi), zwracają wartość logiczną. Jednak operatory && i || faktycznie zwracają wartość jednego z określonych operandów, więc jeśli te operatory są używane z wartościami innymi niż Boolean, mogą zwracać wartość inną niż Boolean. Operatory logiczne są opisane w tabeli powyżej.

```
Welcome                                   Przykła
var a1 =  true && true;
var a2 =  true && false;
var a3 = false && true;
var a4 = false && (3 == 4);
var a5 = 'Kot' && 'Pies';
var a6 = false && 'Kot';
var a7 = 'Kot' && false;
console.log('a1 = ' + a1)
console.log('a2 = ' + a2)
console.log('a3 = ' + a3)
console.log('a4 = ' + a4)
console.log('a5 = ' + a5)
console.log('a6 = ' + a6)
console.log('a7 = ' + a7)

JavaScript - hello.js:9 ✓

a1 = true
a2 = false
a3 = false
a4 = false
a5 = Pies
a6 = false
a7 = false
[Finished in 1.035s]
```

Oto kilka przykładów w praktyce dla operatora logicznego && (ORAZ lub I)

```
Welcome                                   Prz
var a1 =  true || true;
var a2 = false || true;
var a3 =  true || false;
var a4 = false || (3 == 4);
var a5 = 'Kot' || 'Pies';
var a6 = false || 'Kot';
var a7 = 'Kot' || false;
console.log('a1 = ' + a1)
console.log('a2 = ' + a2)
console.log('a3 = ' + a3)
console.log('a4 = ' + a4)
console.log('a5 = ' + a5)
console.log('a6 = ' + a6)
console.log('a7 = ' + a7)

JavaScript - hello.js:5 ✓

a1 = true
a2 = true
a3 = true
a4 = false
a5 = Kot
a6 = Kot
a7 = Kot
[Finished in 1.013s]
```

Oto kilka przykładów w praktyce dla operatora logicznego || (LUB):

```
Welcome
var a1 = !true;
var a2 = !false;
var a3 = !'Kot';

console.log('a1 = ' + a1)
console.log('a2 = ' + a2)
console.log('a3 = ' + a3)


JavaScript - hello.js:1 ✓

a1 = false
a2 = true
a3 = false
[Finished in 1.322s]
```

Oto kilka przykładów w praktyce dla operatora logicznego ! (NEGACJA)

Instrukcja warunkowa if

Bardzo często w trakcie tworzenia kodu skryptu okazuje się, że będziemy musieli skorzystać z instrukcji warunkowych if (jeżeli); są to na przykład takie sytuacje:

- użytkownik może uzyskać dostęp do strony, jeśli ma odpowiedni poziom dostępu,
- sprzedaż produktu jest możliwa, jeśli jest on na stanie magazynowym,
- dane pole ma być podświetlone zawsze wtedy, gdy użytkownik nie wprowadzi w nie żadnej treści.

Aby w kodzie zrobić coś pod określonym warunkiem, używamy instrukcji warunkowej if. Zaczynamy ją od słowa kluczowego if, po czym umieszczamy nawias, w którym znajdzie się warunek. Warunek to wyrażenie, które zwraca wartość logiczną (czyli true lub false). Można go utworzyć na przykład przy użyciu znaku porównania **==**, **<** czy **!=**. Za nawiasami z warunkiem w nawiasach klamrowych określamy, jakie instrukcje mają zostać wykonane, gdy warunek jest spełniony (zwraca true). Jeśli nie jest spełniony (zwraca false), to instrukcje nie zostaną wykonane.

W przykładzie **A** widzimy, że w przypadku, gdy wynik sprawdzenia zwraca wartość logiczną true, wykonywany jest kod wewnątrz instrukcji if, natomiast jeśli nie, po prostu wychodzimy z instrukcji i wykonywany jest dalszy kod. Alternatywnie, tak jak w naszym przykładzie, możemy użyć słowa kluczowego **else**, które przy instrukcji warunkowej id pozwala sformułować operację dla zwróconej przez test logiczny wartości false.

Tak więc do zapamiętania jest informacja, że gdy porównujemy ze sobą dwie wartości (tak jak powyżej zmienną nr z liczbą 55), wynikiem takiej operacji zawsze będzie wartość Boolean, czyli true lub false (prawda lub fałsz) **B**.

Warto też zwrócić uwagę, że zgodnie z tym, czego dowiedzieliśmy się wcześniej o typach danych, takie sprawdzenie, jak w tym przykładzie **C**, zadziała bez problemu, gdyż mimo że sprawdzamy wartość liczbową oraz tekstową, typy zmiennych są dynamicznie dostosowywane – dzięki temu sprawdzenie zadziała poprawnie. Na pewno nie powiodłoby się to w innych językach programowania bez takiej obsługi typów zmiennych.

A

Welcome | Przyklad.html

```
const nr = 55;

if (nr > 30) {
    console.log("Liczba jest większa od 30");
} else {
    console.log("Liczba jest mniejsza lub równa 30");
}
```

JavaScript - hello.js:2 ✓

```
Liczba jest większa od 30
[Finished in 1.04s]
```

B

Welcome | Przyklad.h

```
const a = 20;
const b = 30;

console.log(b > a);
console.log(b < a);
console.log(b === a);

if (a < b) {
    console.log("A jest mniejsze od B");
}
```

JavaScript - hello.js:10 ✓

```
true
false
false
A jest mniejsze od B
[Finished in 1.013s]
```

C

Welcome | Przyklad.h

```
if ("13" > 5) {
    console.log(`Liczba 13 jest większa od 5`);
}
```

JavaScript - hello.js:4 ✓

```
Liczba 13 jest większa od 5
[Finished in 1.015s]
```

```
const animal = "koń"

switch (animal) {
    case "pies":
        console.log("Psy opcja 1");
        break;
    case "kot":
        console.log("Kot opcja 2");
        break;
    case "koń":
        console.log("Koń opcja 3");
        break;
    default:
        console.log("Inny zwierzak");
}
```

```
JavaScript - hello.js:14

Koń opcja 3
[Finished in 1.296s]
```

SWITCH – CASE

Konstrukcja switch jest dość często spotykana w językach programowania. Jest kolejnym sposobem tworzenia warunków – tym razem na zasadzie przyrównania wyniku do konkretnych wartości.
Każdy przypadek kończy się słowem **break**, które kończy wykonywanie instrukcji **switch**.
Jeżeli pominiemy to słowo, wtedy nawet przy pomyślnym przyrównaniu zostaną wykonane kolejne sprawdzenia, co często może powodować błędy. Dodatkowo instrukcja **switch** ma specjalny przypadek **default**, który będzie wybierany, gdy wszystkie inne przypadki będą błędne (odpowiednik else w instrukcji if).

Możemy również skorzystać z bardziej złożonej konstrukcji **else if**, dzięki temu w jednym bloku możemy sprawdzić wiele różnych warunków jeden po drugim, a dopiero po kilku warunkach, gdy wszystkie logiczne testy wypadną negatywnie, będziemy mogli wykonać instrukcje z sekcji **else**.

```
const name = "Test";

if (name === "TEST1") {
    console.log("TEST1");
} else if (name === "TEST2") {
    console.log("TEST2");
} else if (name === "TEST3") {
    console.log("TEST3");
} else {
    console.log("Nie znaleziono odpowiedniego testu");
}
```

```
JavaScript - hello.js:10

Nie znaleziono odpowiedniego testu
[Finished in 1.044s]
```

Pętle i iteracje

Pętle oferują szybki i łatwy sposób na powtarzanie danych. Możemy myśleć o pętli jak o skomputeryzowanej wersji gry, w której mówimy komuś, aby zrobił X kroków w jednym kierunku, a następnie Y w drugim. Jako pętlę można wyrazić na przykład polecenie „Idź pięć kroków na wschód” **A**.
Istnieje wiele różnych rodzajów pętli, ale zasadniczo wszystkie robią to samo – po-

A

```
for (let krok = 0; krok < 5; krok++) {
  // Idź 5 kroków, wartości kroków od 0 do 4.
  console.log('Idź jeden krok na wschód');
}
```

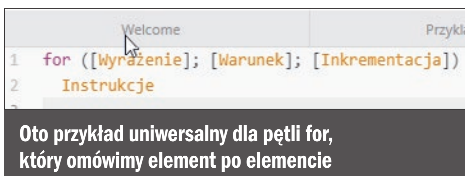
```
JavaScript - hello.js:4

Idź jeden krok na wschód
Idź jeden krok na wschód
Idź jeden krok na wschód
Idź jeden krok na wschód
Idź jeden krok na wschód
[Finished in 1.199s]
```

wtarzają czynność kilka razy. (Warto pamiętać, że liczba może wynosić zero).
Różne mechanizmy pętli oferują różne sposoby określania punktu początkowego i końcowego pętli. Istnieją różne sytuacje, w których jeden typ pętli jest łatwiejszy w implementacji niż inne.

Pętla for

Pętla for powtarza się, dopóki określony warunek nie zmieni się na fałsz. Pętla for w JavaScripcie jest podobna do pętli for w Javie i C.



Oto przykład uniwersalny dla pętli for, który omówimy element po elemencie

1 Wyrażenie inicjujące **[Wyrażenie]**, jeśli istnieje, jest wykonywane. To wyrażenie zwykle inicjuje jeden lub więcej liczników pętli, ale składnia umożliwia wyrażenie o dowolnym stopniu złożoności. To wyrażenie może również deklarować zmienne.

2 Wyrażenie **[Warunek]** jest testem logicznym. Jeśli wartość **[Warunek]** jest true, instrukcje pętli są wykonywane. W przeciwnym razie pętla for się kończy. (Jeżeli wyrażenie **[Warunek]** zostanie całkowicie pominięte, przyjmuje się, że warunek jest prawdziwy).

3 **Instrukcje**. Aby wykonać wiele instrukcji, używamy instrukcji blokowej (**{ ... }**), aby je pogrupować.

4 Jeśli zadeklarujemy inkrementację, wykonywane jest wyrażenie w obszarze **[Inkrementacja]**.

5 Pętla wraca do kroku **2**.

To teoria – a teraz przyjrzymy się znacznie bardziej złożonemu przykładowi szkoleniowemu, który będzie zawierać większość elementów omówionych na poprzednich stronach.
W tym przykładzie jest instrukcja **for**, która służy do zliczania, ile konkretnie elementów wybraliśmy na liście w dokumencie HTML.
Oto kod dokumentu HTML, który musimy umieścić w sekcji **<body>** **A**.
Teraz napiszemy skrypt JS, który pozwoli na zliczenie zaznaczonych elementów na formularzu **B** (patrz następna strona).
W pętli **for** deklarujemy zmienną **i** z wartością 0, sprawdzamy, czy **i** ma mniejszą wartość niż liczba opcji zaznaczonych w formularzu i wewnątrz pętli zwiększamy o 1 wartość zmiennej przechowującej informację o zaznaczonych elementach.
W kodzie znajduje się również funkcja oraz kilka innych bardziej zaawansowanych elementów. Zapisujemy dokumenty i uruchamiamy stronę.

A

```
<body>
<form name="selectForm">
  <label for="typMuzyki">Wybierz typ muzyki, który lubisz</label>
  <select id="typMuzyki" name="typMuzyki" multiple>
    <option selected>Pop</option>
    <option>Jazz</option>
    <option>Blues</option>
    <option>New Age</option>
    <option>Classical</option>
    <option>Opera</option>
  </select>
  <button id="btn" type="button">Ile jest wybranych?</button>
</form>
```

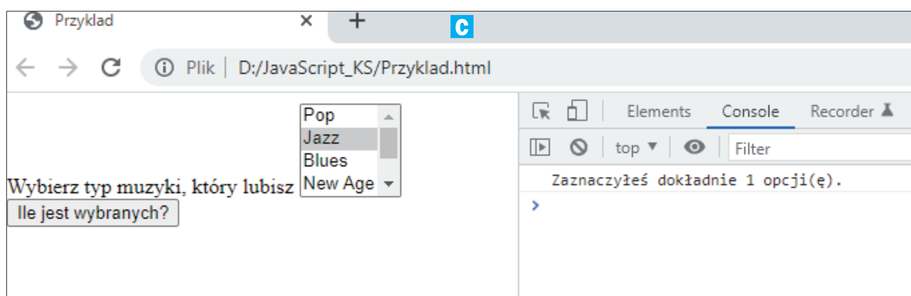
B

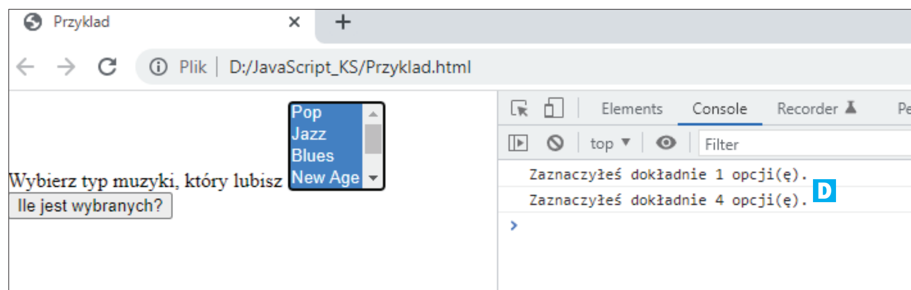
```
function jakDuzo(wskazObiekt) {
  let ileZaznaczonych = 0;
  for (let i = 0; i < wskazObiekt.options.length; i++) {
    if (wskazObiekt.options[i].selected) {
      ileZaznaczonych++;
    }
  }
  return ileZaznaczonych;
}

const btn = document.getElementById('btn');

btn.addEventListener('click', () => {
  const typMuzyki = document.selectForm.typMuzyki;
  console.log(`Zaznaczyłeś dokładnie ${jakDuzo(typMuzyki)} opcji(ę).`);
});
```

Po uruchomieniu strony zobaczymy formularz C. Możemy klikać na poszczególne dostępne w nim opcje. W celu zaznaczenia więcej niż jednego elementu musimy trzymać wciśnięty klawisz ctrl.
Klikamy na przycisk **Ile jest wybranych?**. Odpowiedź będziemy mogli odczytać w konsoli. Wciskamy więc kombinację klawiszy ctrl+shift+J i z konsoli odczytujemy odpowiedź.
Po wybraniu większej liczby elementów i kliknięciu na **Ile jest wybranych?** za każdym razem zostanie wyświetlony poprawny wynik D. Jest to możliwe dzięki pętli **for**.





Pętla do ... while

Pętla **do ... while** wykonuje się, dopóki podany warunek wyjścia nie będzie miał wartości false.

```
Welcome
do
  Instrukcje
while (Warunek);
```

W przypadku tego rodzaju pętli instrukcje zostaną wykonane zawsze przynajmniej raz, zanim warunek wyjścia zostanie sprawdzony. Jeśli warunek będzie miał wartość logiczną **true**, pętla będzie wykonywana dalej, a jeżeli **false**, zakończy swoje działanie. Zatem warunek sprawdzany jest na samym końcu i zawsze instrukcje zostają wykonane przynajmniej raz.

```
let i = 0;
do {
  i += 1;
  console.log(i);
} while (i < 6);

JavaScript - hello.js:5 ✓
1
2
3
4
5
6
[Finished in 1.286s]
```

Jak widać w tym przykładzie, instrukcje wewnątrz bloku były wykonywane, dopóki warunek końcowy nie zwrócił fałszu

```
let i = 0;
do {
  i += 1;
  console.log(i);
} while (i == 100);

JavaScript - hello.js:6 ✓
1
[Finished in 1.027s]
```

Należy zawsze pamiętać przy tego typu pętli, że nawet gdy wynik testu logicznego od samego początku to fałsz, instrukcje zostaną wykonane jeden raz

Pętla while

W przeciwieństwie do pętli do … while, w przypadku pętli **while** test logiczny wykonywany jest już na wejściu do pętli. Jeśli więc zwróci on fałsz, nie zostanie wykonana żadna instrukcja wewnątrz pętli.

```
let n = 0;
let x = 0;
while (n < 3) {
  n++;
  x += n;
  console.log(n)
}

JavaScript - hello.js:6 ✓
1
2
3
[Finished in 1.036s]
```

Warunek wstępny został spełniony, więc pętla została zrealizowana i dopiero, gdy inkrementacja zwiększyła odpowiednio wartość n, opuściliśmy pętlę

```
Welcome
let n = 100;
let x = 0;
while (n < 10) {
  n++;
  x += n;
  console.log(n)
}

JavaScript - hello.js:7 ✓
[Finished in 1.013s]
```

W przypadku pętli while, gdy warunek wstępny nie zostanie spełniony, instrukcje nie zostaną nigdy wykonane

Te podstawowe pętle i słowa kluczowe wystarczą nam, abyśmy mogli pisać pierwsze skrypty, musimy jednak jeszcze poznać funkcje.

Welcome

```
while (true) {
  console.log('Nieskończoność!');
}
```

JavaScript - hello.js:4

```
Nieskończoność!
Nieskończoność!
Nieskończoność!
Nieskończoność!
Nieskończoność!
Nieskończoność!
Nieskończoność!
Nieskończoność!
Nieskończoność!
Nieskończoność!
Nieskończoność!
Nieskończoność!
Nieskończoność!
ieskończoność!
eskończoność!
ieskończoność!
Nieskończoność!
Nieskończoność!
Nieskończoność!
Nieskończoność!
Nieskończoność!
Nieskończoność!
```

UWAGA NA NIESKOŃCZONE PĘTLE

Istnieje w programowaniu pojęcie nieskończonej pętli – taka pętla to coś, czego zdecydowanie powinniśmy za wszelką cenę unikać. W zależności od programu wejście w taką pętlę może doprowadzić do zakończenia działania programu, a nawet do ponownego uruchomienia komputera.
Jest to klasyczny przykład nieskończonej pętli – pętla będzie wykonywana, dopóki warunek true nie zmieni wartości na false, jednak kod instrukcji wewnątrz pętli nie umożliwia w żaden sposób wyjścia z takiej pętli.
Uwaga! Zanim przetestujemy działanie tego przykładu na naszej maszynie, koniecznie zapiszmy wszystkie ważne dokumenty, gdyż może okazać się, że będzie konieczny restart komputera.

INSTRUKCJA BREAK

Czasem pętle, które mogą być wykonywane teoretycznie w nieskończoność, są przydatne – na przykład do tego, żeby bez przerwy sprawdzać jakąś wartość. Możemy w takim wypadku skorzystać z instrukcji break, aby natychmiast opuścić taką pętlę.

```
let x = 0;
let z = 0;
while (true) {
  console.log('Iteracja pętli: ' + x);
  x += 1;
  z += 1;
    if (z === 5 && x === 5) {
      console.log('Opuszczam pętlę');
      break;
    }
}
```

JavaScript - hello.js:8 ✓

```
Iteracja pętli: 0
Iteracja pętli: 1
Iteracja pętli: 2
Iteracja pętli: 3
Iteracja pętli: 4
Opuszczam pętlę
[Finished in 1.025s]
```

Funkcje

We wcześniej podawanych przykładach używaliśmy już funkcji, teraz dowiemy się dokładniej, czym są i jak ich można używać.
Zacznijmy od tego, że funkcję należy zdefiniować lub zadeklarować. Zawsze odbywa się to w ściśle określony sposób. Na początku używamy słowa kluczowego **function**, potem umieszczamy nazwę funkcji – korzystając z tej nazwy, będziemy wywoływać funkcję w dalszej części skryptu. Następnie w nawiasie zamieszczamy listę parametrów funkcji, które

oddzielamy przecinkami. Na koniec w nawiasie klamrowym podajemy instrukcje, które mają być zrealizowane po wywołaniu funkcji.

A

```
function kwadrat(liczba) {
  return liczba * liczba;
}
var x,y;
x = 5;
y = kwadrat(x);
console.log("x do kwadratu to : " + y);
```

JavaScript - hello.js:7 ✓

```
x do kwadratu to : 25
[Finished in 1.013s]
```

Welcome

```
function silnia(n) {
  if ((n === 0) || (n === 1))
    return 1;
  else
    return (n * silnia(n - 1));
}

console.log(silnia(5));
```

JavaScript - hello.js:8 ✓

```
120
[Finished in 1.095s]
```

Funkcja może wywołać samą siebie, jak na przykład ta funkcja, która rekurencyjnie oblicza silnię

Oto prosty przykład funkcji obliczającej kwadrat podanej liczby **A**. Funkcja **kwadrat** przyjmuje tylko jeden parametr. Wewnątrz funkcji mamy jedną instrukcję, która składa się ze słowa kluczowego **return**. Służy ono do zwracania konkretnej wartości funkcji. Parametr funkcji jest traktowany jako zmienna lokalna. Funkcję wywołujemy, podając w kodzie jej nazwę oraz przekazując dokładną liczbę parametrów, które są jej przypisane. Tak więc w naszym przykładzie po wywołaniu funkcji z parametrem **x**, który ma wartość **5**, wewnątrz funkcji ta wartość jest przypisana do parametru **liczba**, wykonane są instrukcje i poprzez słowo klucz **return** zwrócona jest wartość funkcji. Ta wartość w kodzie przypisana jest poza funkcją do zmiennej **y**.

Gra w zgadywanie liczby

W celu ugruntowania wszystkich wiadomości zdobytych w tym rozdziale, napiszemy kod gry w JavaScripcie, w którą będziemy mogli grać w oknie przeglądarki. Podejdziemy do tego zadania tak jak programista – najpierw zapoznamy się z zadaniem, podzielimy je na etapy i przejdziemy do realizacji.

Musimy napisać prostą grę typu „zgadnij liczbę". Gra powinna wybierać losową liczbę między 1 a 100. Zadaniem gracza jest odgadnąć tę liczbę w najwyżej 10 próbach. Po każdej próbie gracz powinien otrzymać informację, czy zgadł, czy też nie i – jeśli nie odgadł – powinien dodatkowo dowiedzieć się, czy liczba, którą podał, miała wartość za małą, czy za dużą. Ponadto gracz powinien widzieć wybrane poprzednio przez siebie liczby. Gra ma się zakończyć, gdy gracz poda prawidłową odpowiedź lub gdy wykorzysta ostatnią próbę. Po zakończeniu gry gracz powinien móc rozpocząć ją od nowa.

Bardzo często możemy się spotkać z podobnym lub mniej precyzyjnym opisem zadania. Bardzo ważne jest w wypadku realizacji tego typu zadań to, aby już na samym początku jasno określić wszelkie wymagania – dzięki temu będziemy mogli dość dokładnie oszacować czasochłonność i opłacalność realizacji. Warto rozbić taką treść na proste kroki i – można powiedzieć – przekształcić na schemat blokowy.

A

Welcome | Przyklad.html | zgadywanie_liczby.html

```
<!DOCTYPE html>
<html>
  <head>
    <meta charset="utf-8">
    <title>Zgadnij liczbę</title>
    <style>
      html {
        font-family: sans-serif;
      }

      body {
        width: 50%;
        max-width: 800px;
        min-width: 480px;
        margin: 0 auto;
      }
      .lastResult {
        color: white;
        padding: 3px;
      }
    </style>
  </head>
  <body>
    <h1>Zgadnij liczbę</h1>
    <p>Program wybrał liczbę od 1 do 100. Sprawdź, czy uda Ci się ją odgadnąć w
      mniej niż 10 prób. Otrzymasz podpowiedzi o tym, czy zgdywana przez Ciebie
        wartość jest zbyt dsuza lub zbyt mała</p>
    <div class="form">
      <label for="guessField">Wprowadź liczbę: </label>
      <input type="text" id="guessField" class="guessField">
      <input type="submit" value="Submit guess" class="guessSubmit">
    </div>
    <div class="resultParas">
      <p class="guesses"></p>
      <p class="lastResult"></p>
      <p class="lowOrHi"></p>
    </div>
    <script type="text/javascript" src="zgadywanie.js"></script>
  </body>
</html>
```

JavaScript - hello.js:8 ✓

Schemat gry

1 Wybierz losową liczbę z zakresu od 1 do 100.

2 Zapisz numer próby, którą podejmuje gracz. Zacznij od 1.

3 Podaj graczowi sposób, w jaki może odgadnąć tę liczbę.

4 Gdy padnie odpowiedź, zapisz ją, aby użytkownik mógł widzieć swoje poprzednie próby.

5 Sprawdź, czy padła prawidłowa odpowiedź.

6 Jeśli tak:
- **A** Wyświetl gratulacje.
- **B** Zablokuj możliwość podawania dalszych odpowiedzi (to mogłoby namieszać w grze).
- **C** Udostępnij narzędzie, którym gracz może ponownie uruchomić grę.

7 Jeśli nie padła prawidłowa odpowiedź i graczowi pozostały jeszcze próby:
- **A** Poinformuj o nieprawidłowej odpowiedzi.
- **B** Pozwól podać kolejną odpowiedź.
- **C** Zwiększ numer próby gracza o 1.

8 Jeśli nie padła prawidłowa odpowiedź i graczowi nie pozostała już ani jedna próba:
- **A** Poinformuj o zakończeniu gry.
- **B** Zablokuj możliwość podawania dalszych odpowiedzi (to mogłoby namieszać w grze).
- **C** Udostępnij narzędzie, którym gracz może ponownie uruchomić grę.

9 Gdy gra uruchomi się ponownie, upewnij się, że dane z poprzedniej gry zostały całkowicie usunięte i interfejs powrócił do stanu początkowego. Przejdź do punktu **1**.

Zaczynamy pracę nad grą

Mając taką rozpiskę, możemy przystąpić do zamiany jej na kod źródłowy, który zrealizuje nasze pomysły.

W celu ułatwienia sobie zadania warto pobrać z **KŚ+** (**ksplus.pl**) gotowy kod HTML strony dla gry – **zgadywanie_liczby.html** **A**.

Skrypt JS należy umieścić w tym samym folderze co plik HTML i nadać mu nazwę **zgadywanie.js**.

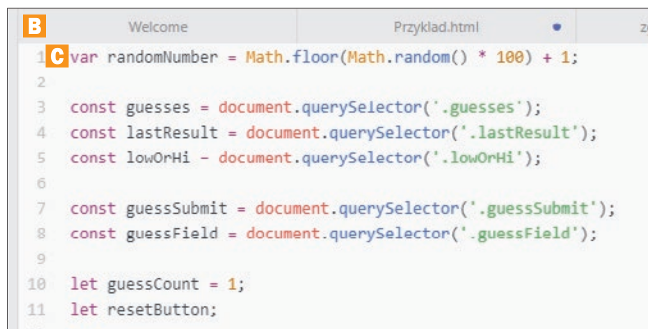
Otwieramy plik JS w Atomie i jednocześnie plik HTML w przeglądarce. Plik ten zawiera nagłówek, akapit z instrukcją gry i (jeszcze niedziałający) formularz do wprowadzania odpowiedzi.



Zaczynamy od przygotowania sobie wszystkich zmiennych, z jakich będziemy korzystać. Ta część kodu **B** definiuje zmienne i stałe niezbędne do pracy programu.

Zmienne są pojemnikami na wartości takie jak liczby czy ciągi znaków.

Zmienną tworzymy, używając słowa kluczowego **let** (lub **var**), po którym wpisujemy nazwę tej zmiennej.




```
var randomNumber = Math.floor(Math.random() * 100) + 1;

const guesses = document.querySelector('.guesses');
const lastResult = document.querySelector('.lastResult');
const lowOrHi = document.querySelector('.lowOrHi');

const guessSubmit = document.querySelector('.guessSubmit');
const guessField = document.querySelector('.guessField');

let guessCount = 1;
let resetButton;
```

Następnie możemy tej zmiennej przypisać wartość. Robimy to za pomocą znaku równości (**=**), po którego prawej stronie wpisujemy wartość.

Stałe natomiast mają za zadanie przechować dane, które mają się nie zmieniać. Stałe tworzy się podobnie

jak zmienne, ale przy użyciu słowa kluczowego **const**. W naszym przykładzie użyjemy stałych do przechowania odnośników (referencji) do poszczególnych części naszego interfejsu użytkownika. Tekst w niektórych z nich może w którymś momencie ulec zmianie, jednak bloki kodu HTML, do których odnoszą się nasze stałe, pozostaną niezmienne.

W naszym przykładzie

- Pierwsza zmienna – **randomNumber** C – ma przypisaną losową liczbę z zakresu od 1 do 100 wybraną przez matematyczny algorytm.
- Każda z trzech kolejnych stałych zawiera referencje do konkretnych paragrafów w kodzie

D
```
<div class="resultParas">
  <p class="guesses"></p>
  <p class="lastResult"></p>
  <p class="lowOrHi"></p>
</div>
```

HTML. Zostaną one użyte do wstawienia odpowiednich wartości w dalszej części kodu D.
- Następne dwie zmienne mają za zadanie przechować referencje do pola tekstowego i przycisku zatwierdzania odpowiedzi. Przydadzą się do wprowadzania i zatwierdzania kolejnych odpowiedzi gracza E.

E
```
<div class="form">
  <label for="guessField">Wprowadź liczbę: </label>
  <input type="text" id="guessField" class="guessField">
  <input type="submit" value="Wyślij odpowiedź" class="guessSubmit">
</div>
```

- Dwie ostatnie zmienne zawierają kolejno: wartość domyślną 1 (tej zmiennej użyjemy do liczenia prób odpowiedzi) oraz referencję do przycisku „reset" (co prawda jeszcze nie istnieje, ale niebawem się to zmieni).

Ponieważ zamierzamy wykonywać różnego typu operacje, będziemy potrzebować funkcji, która będzie sprawdzała każdą odpowiedź gracza i w odpowiedni sposób reagowała.
Jest to już zdecydowanie bardziej rozbudowana funkcja F, która zawiera przykład wykorzystania instrukcji warunkowych. Omówimy ją krok po kroku.

F
```
function checkGuess() {
  var userGuess = Number(guessField.value);
  if (guessCount === 1) {
    guesses.textContent = 'Poprzednio wprowadzone liczby: ';
  }
  guesses.textContent += userGuess + ' ';

  if (userGuess === randomNumber) {
    lastResult.textContent = 'Gratulacje! Zgadłeś';
    lastResult.style.backgroundColor = 'green';
    lowOrHi.textContent = '';
    setGameOver();
  } else if (guessCount === 10) {
    lastResult.textContent = '!!!Koniec gry!!!';
    setGameOver();
  } else {
    lastResult.textContent = 'Źle!';
    lastResult.style.backgroundColor = 'red';
    if(userGuess < randomNumber) {
      lowOrHi.textContent = 'Zbyt mała liczba!';
    } else if(userGuess > randomNumber) {
      lowOrHi.textContent = 'Zbyt duża liczba!';
    }
  }

  guessCount++;
  guessField.value = '';
  guessField.focus();
}

JavaScript - hello.js:8 ✓
```

- Pierwsza linia w ciele funkcji deklaruje zmienną **userGuess** i przypisuje jej wartość równą obecnej wartości wpisanej do pola tekstowego. Jednocześnie weryfikujemy tę wartość wbudowaną metodą **Number()**, aby mieć pewność, że wartość ta jest na pewno liczbą.

■ Następnie jest nasz pierwszy blok kodu z instrukcją warunkową. Instrukcja warunkowa pozwala nam uruchamiać inne części kodu w zależności od tego, czy dany warunek zostanie spełniony, czy też nie. Najprostsza instrukcja warunkowa rozpoczyna się słowem kluczowym if, po którym następują nawiasy zwykłe i dalej nawiasy klamrowe. Wewnątrz nawiasów zwykłych umieszczamy nasz warunek.
Jeśli warunek jest spełniony (wartość true), uruchamiany jest kod wewnątrz nawiasów klamrowych. Jeżeli nie (wartość false) – kod w nawiasach klamrowych nie jest uruchamiany i przechodzimy do następnej części kodu. W tym przypadku następuje sprawdzenie, czy zmienna **guessCount** jest równa 1 (to znaczy czy jest to pierwsza odpowiedź gracza). Jeśli tak jest, zmieniamy tekst paragrafu (akapitu), w przeciwnym wypadku – nie.

```
13  function checkGuess() {
14    var userGuess = Number(guessField.value);
15    if (guessCount === 1) {
16      guesses.textContent = 'Poprzednio wprowadzone liczby: ';
17    }
18 G  guesses.textContent += userGuess + ' ';
```

■ Linia 18 G dodaje aktualną wartość zmiennej userGuess na końcu akapitu **guesses** oraz białą spację, aby oddzielić od siebie kolejne odpowiedzi.

■ Następna część kodu (linie 20–36) wykonuje kilka operacji sprawdzenia:

- Pierwsza instrukcja **if(){ }** sprawdza, czy odpowiedź gracza jest równa wartości zmiennej **randomNumber** wybranej losowo na początku naszego programu. Jeśli tak jest, oznacza to, że gracz odpowiedział poprawnie i gra jest zakończona. Możemy zatem pogratulować zwycięzcy, używając zielonego koloru, wyczyścić zawartość akapitu informującego, czy podana liczba jest za mała lub za duża (paragraf **lowOrHi**) i uruchomić funkcję **setGameOver()**, o której dowiemy się więcej za chwilę.

```
if (userGuess === randomNumber) {
  lastResult.textContent = 'Gratulacje! Zgadłeś';
  lastResult.style.backgroundColor = 'green';
  lowOrHi.textContent = '';
  setGameOver();
```

- Bezpośrednio pod właśnie opisaną instrukcją rozpoczynamy kolejny test, używając struktury **else if(){ }**. Sprawdza ona, czy obecna próba odpowiedzi gracza jest ostatnią (równą 10). Jeśli tak jest, program wykona te same operacje co poprzednio, z tą różnicą, że zamiast gratulacji ma wyświetlić napis „Koniec gry".

```
} else if (guessCount === 10) {
  lastResult.textContent = '!!!Koniec gry!!!';
  setGameOver();
```

- Ostatnia z instrukcji warunkowych ma postać **else { }** i zawiera kod uruchamiany tylko w przypadku, gdy żaden z poprzednich warunków nie został spełniony (czyli gdy gracz nie odgadł liczby, ale pozostały mu jeszcze próby). W tym przypadku informujemy go, że się pomylił i przeprowadzimy kolejne sprawdzenie, czy wpisana przez niego liczba była większa czy mniejsza od prawidłowej odpowiedzi. Oczywiście wyświetlony zostaje odpowiedni komunikat.

```
} else {
  lastResult.textContent = 'Źle!';
  lastResult.style.backgroundColor = 'red';
  if(userGuess < randomNumber) {
    lowOrHi.textContent = 'Zbyt mała liczba!';
  } else if(userGuess > randomNumber) {
    lowOrHi.textContent = 'Zbyt duża liczba!';
  }
}
```

■ Trzy ostatnie linie naszej funkcji **checkGuess(){ }** (linie 38–40) przygotowują grę do przyjęcia kolejnej odpowiedzi. W tym celu zostaje dodane 1 do wartości zmiennej **guessCount**, aby „zużyć" jedną próbę gra-

```
38      guessCount++;
39      guessField.value = '';
40      guessField.focus();
41    }
```

cza (**++** oznacza zwiększenie o 1 – inkrementację), wyczyszczone zostaje pole formularza, aby gracz wygodnie mógł wprowadzić swoją następną odpowiedź, i ustawiony zostaje w tym polu oczekujący kursor.

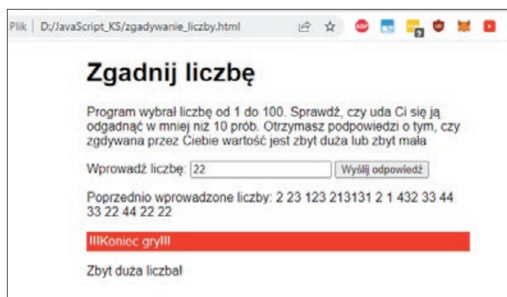
Obsługa zdarzeń

Mimo że poprawnie zadeklarowaliśmy funkcję **checkGuess()**, póki co nie wykona ona żadnej akcji. Powód jest bardzo prosty – nie mamy miejsca w kodzie, gdzie moglibyśmy wywołać naszą funkcję. Z naszych ustaleń wynika, że funkcja ta ma zostać wywołana po naciśnięciu przycisku Wyślij odpowiedź **na stronie w przeglądarce**. Więc musimy wywołanie funkcji zrealizować z wykorzystaniem **zdarzenia (event)**. Zdarzenie jest tym, co dzieje się w przeglądarce (na przykład kliknięcie przycisku, załadowanie strony, odtwarzanie filmu itd.) i czego możemy użyć w celu wywołania konkretnego bloku kodu. Konstrukty, które „nasłuchują", czy miało miejsce zdarzenie, nazywane są detektorami zdarzeń (ang. **event listeners**), a wywoływane w odpowiedzi na nie bloki kodu – modułami obsługi zdarzeń (ang. **event handlers**). Do naszego skryptu poza funkcją dodajemy linię:

```
guessSubmit.addEventListener('click', checkGuess);
```

W ten sposób dodaliśmy detektor zdarzenia do przycisku guessSubmit. Jest to metoda, która ma dwie dane wejściowe (zwane argumentami) zapisane w formie ciągu znaków: typ zdarzenia, które ma zajść (w tym przypadku click – czyli kliknięcie kursorem), oraz fragment kodu, który ma zostać uruchomiony poprzez to zdarzenie (funkcja checkGuess()). Warto również zwrócić uwagę, że nasza funkcja checkGuess nie ma żadnych parametrów wejściowych, więc możemy ją wywołać, podając tylko jej nazwę, pomijając pusty nawias.

Zapisujemy i odświeżamy swój kod. Powinien już prawie w pełni działać. Pozostała jeszcze jedna kwestia: gdy odgadniemy właściwą odpowiedź lub wykorzystamy wszystkie próby odpowiedzi, gra powinna zostać przerwana, ponieważ jak dotąd nie zdefiniowaliśmy funkcji setGameOver(), która ma zostać wywołana w przypadku zakończenia gry – tak się nie stanie. Dodajmy zatem brakującą część kodu, aby nasza gra zyskała wszystkie funkcjonalności.



Funkcja setGameOver()

Poniżej naszego detektora zdarzenia zadeklarujemy kolejną funkcję **setGameOver**, która pozwoli na obsługę zakończenia gry i nie tylko.

- Pierwsze dwa wiersze wyłączają wprowadzanie tekstu formularza i przycisk, ustawiając ich właściwość zablokowania na true. Jest to konieczne, ponieważ gdybyśmy tego nie zrobili, użytkownik mógłby przesłać więcej zgadywań po zakończeniu gry, co nie byłoby zgodne z wymaganiami.
- Następne trzy wiersze generują nowy element **<button>**, ustawiają jego etykietę tekstową na „Rozpocznij nową grę!" i dodają go na dole naszego istniejącego kodu HTML.

```
guessSubmit.addEventListener('click', checkGuess);

function setGameOver() {
  guessField.disabled = true;
  guessSubmit.disabled = true;
  resetButton = document.createElement('button');
  resetButton.textContent = 'Rozpocznij nową grę!';
  document.body.appendChild(resetButton);
  resetButton.addEventListener('click', resetGame);
}
```

```
function setGameOver() {
  guessField.disabled = true;
  guessSubmit.disabled = true;
```

■ Ostatnia linia ustawia detektor zdarzeń na naszym nowym przycisku, dzięki czemu po kliknięciu na niego wywoływana jest funkcja **resetGame()**.

```
resetButton = document.createElement('button');
resetButton.textContent = 'Rozpocznij nową grę!';
document.body.appendChild(resetButton);
```

Funkcja resetGame()

W poprzedniej funkcji wywołujemy funkcję resetGame, jednak jeszcze nigdzie jej nie zadeklarowaliśmy – musimy teraz to zrobić

```
resetButton.addEventListener('click', resetGame);
```

i dodać do niej instrukcje, które pozwolą nam obsłużyć poprawnie zakończenie naszej gry.

Ten dość długi blok kodu **A** całkowicie resetuje wszystko do stanu, w jakim było na początku gry, więc gracz może spróbować jeszcze raz. To:

1. Ponownie ustawia **guessCount** do 1.

```
function resetGame() {
  guessCount = 1;
```

2. Czyści wszystkie akapity informacyjne **B**.

3. Usuwa przycisk resetowania z naszego kodu **C**.

4. Włącza elementy formularza oraz czyści i ustawia kursor w polu tekstowym **D**.

5. Usuwa kolor tła z akapitu – **lastResult** **E**.

```
function setGameOver() {                                          A
  guessField.disabled = true;
  guessSubmit.disabled = true;
  resetButton = document.createElement('button');
  resetButton.textContent = 'Rozpocznij nową grę!';
  document.body.appendChild(resetButton);
  resetButton.addEventListener('click', resetGame);
}

function resetGame() {
  guessCount = 1;

  var resetParas = document.querySelectorAll('.resultParas p');
  for (var i = 0 ; i < resetParas.length ; i++) {                  B
    resetParas[i].textContent = '';
  }

  resetButton.parentNode.removeChild(resetButton);  C

  guessField.disabled = false;
  guessSubmit.disabled = false;  D
  guessField.value = '';
  guessField.focus();

  lastResult.style.backgroundColor = 'white';  E

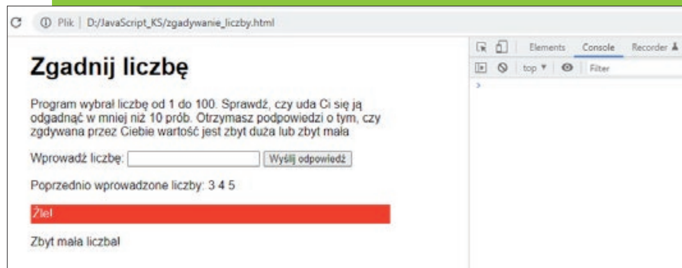
  randomNumber = Math.floor(Math.random() * 100) + 1;
}
```

DODATKOWE ZMIANY W KONSOLI

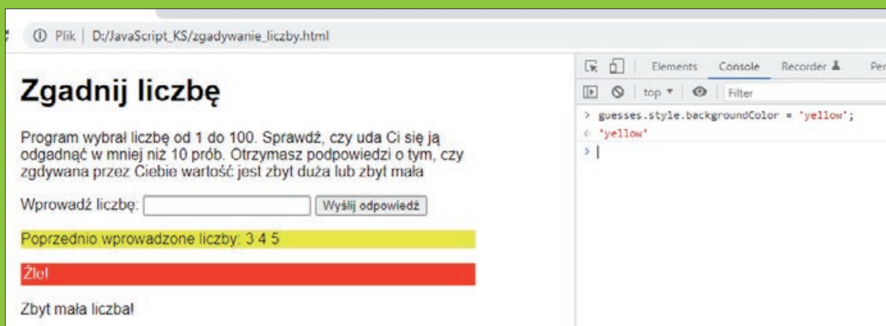
Możemy również na tym etapie pobawić się naszą grą od strony konsoli, wprowadzając różnego typu polecenia, które w trakcie rozgrywki będą zmieniały parametry naszej gry. Możemy zmieniać wygląd całej witryny dzięki temu, że każdy element ma zastosowane style CSS.

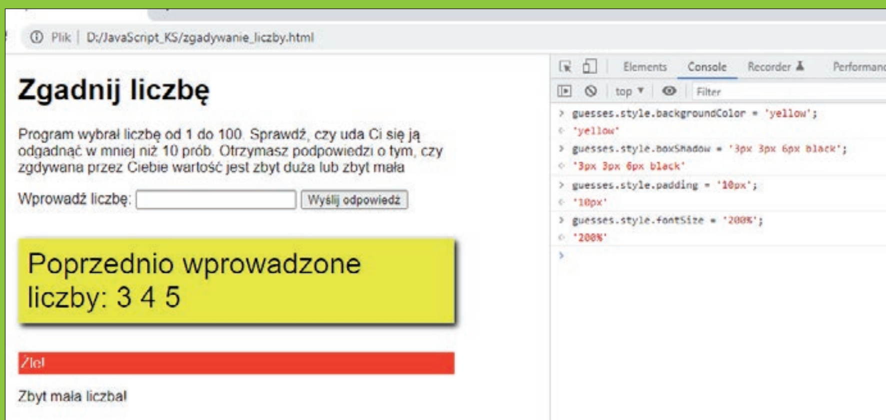
1 Rozpoczynamy naszą grę i otwieramy konsolę ctrl+shift+J.

2 Następnie wprowadzamy polecenie: **guesses.style.backgroundColor = 'yellow';**



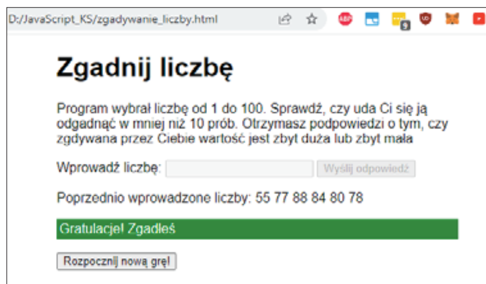
3 Tło poprzednio wprowadzonych liczb zostanie oznaczone kolorem żółtym. W myśl tej zasady możemy dowolnie dostosować wygląd każdego elementu do naszych potrzeb – wystarczy znajomość stylów CSS.

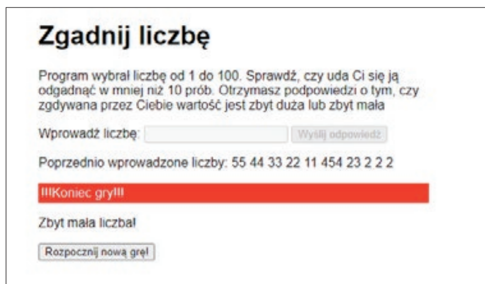




6 Generuje nową liczbę losową **F**, dzięki czemu nie zgadujemy ponownie tej samej liczby.
W tym momencie uzyskujemy w pełni działającą (prostą) grę – gratulacje!

```
    randomNumber = Math.floor(Math.random() * 100) + 1;
}
```
F





OSZUKUJEMY W GRACH

Dzięki temu, że z poziomu konsoli mamy dostęp do całego skryptu i poszczególnych zmiennych, możemy oszukiwać w grach. W ten sam sposób można bez problemu oszukiwać w wielu przeglądarkowych grach.

1 Można oszukiwać na wiele sposobów, po pierwsze wiemy, że gra pozwala 10 razy odgadnąć liczbę – możemy dać sobie jednak więcej szans.

2 Zmienna **guessCount** przechowuje informacje o tym, ile prób wykonaliśmy. Wyświetlamy jej obecną wartość po 3 próbach.

```
<· undefined
> guessCount
<· 4
>
```

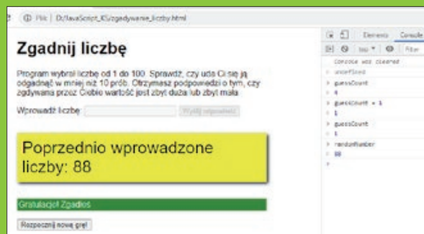
3 Ma ona wartość 4, co jest poprawne, gdyż zaczynamy z poziomu 1. W celu dodania sobie więcej szans wystarczy, że ponownie w trakcie rozgrywki przypiszemy jej wartość 1.

```
> guessCount
<· 4
> guessCount = 1
<· 1
> guessCount
<· 1
> |
```

4 W ten sposób możemy zgadywać, ile tylko razy chcemy. Jeśli jednak zależy nam na jak najszybszym zwycięstwie, wystarczy sprawdzić, jakiej liczby szukamy – przechowywana jest ona w zmiennej **randomNumber**.

```
> randomNumber
<· 88
>
```

5 Odczytaliśmy, że wartość szukanej liczby to 88 – wystarczy podać ją jako rozwiązanie i od razu wygramy.



Oczywiście nie uda nam się w każdej grze i skrypcie tak łatwo wprowadzać modyfikacji, wystarczy, że kod będzie przetwarzany po stronie serwera lub będą zakodowane konkretne wartości i już nie będziemy mogli uzyskać przewagi w prosty sposób.

6 Własna aplikacja webowa w JS

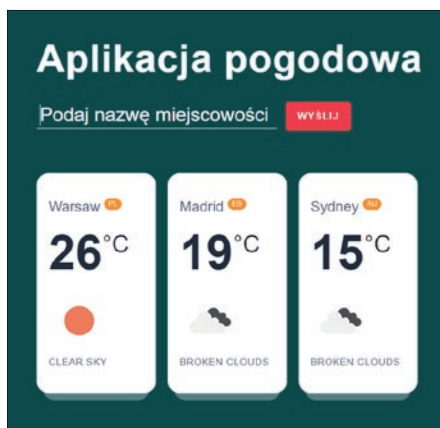
Znając już podstawy, możemy przejść do nieco bardziej zaawansowanych elementów i funkcjonalności JavaScriptu. Zobaczmy krok po kroku, jak stworzyć aplikację webową, która będzie wyświetlała pogodę dla dowolnego miasta na świecie

Własna aplikacja pogodowa

W tym projekcie skupimy się na tym, aby stworzyć w pełni funkcjonalną aplikację webową. Będzie ona miała pole tekstowe – po wpisaniu w nie nazwy dowolnego miasta na świecie otrzymamy informację o aktualnej pogodzie dla tego miasta.

Zaczynamy od określenia w kilku punktach, co chcemy osiągnąć.

- Musimy przygotować dokument HTML, który pozwoli na wyświetlanie niezbędnych danych.
- Musimy dostosować style CSS do naszej aplikacji.
- Musimy dodać kod JavaScript, który będzie uwzględniał:
  - obsługę wprowadzania przez użytkownika danych tekstowych,
  - wyszukiwanie danych pogodowych dla wskazanego przez użytkownika miasta,
  - interpretację danych z serwera zewnętrznego,
  - prezentację tych danych,
  - obsługę błędów – na przykład brak podanego miasta w bazie lub niepodanie wartości typu tekst.



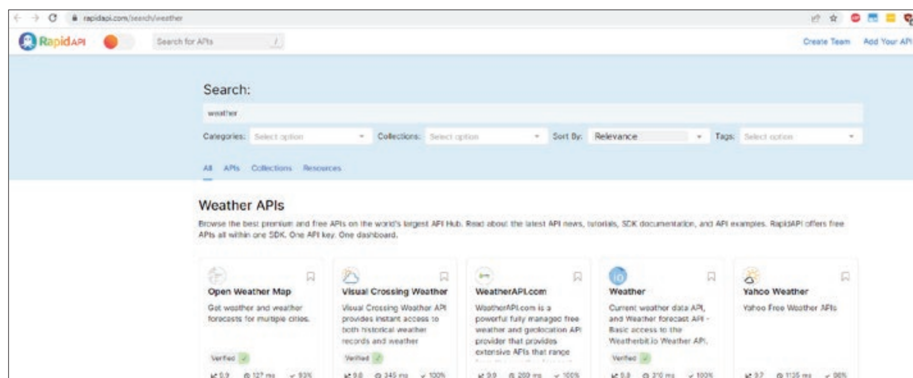
Wszystko do tej pory wygląda dość podobnie do przykładu z grą w zgadywanie z rozdziału piątego, jednak jest kluczowa różnica – musimy pobrać dane pogodowe z serwera zewnętrznego. Zrobimy to za pomocą API i requestu (zapytania) AJAX. Zacznijmy więc od tego pierwszego, a tym drugim zajmiemy się na dalszym etapie projektu.

CZYM JEST API

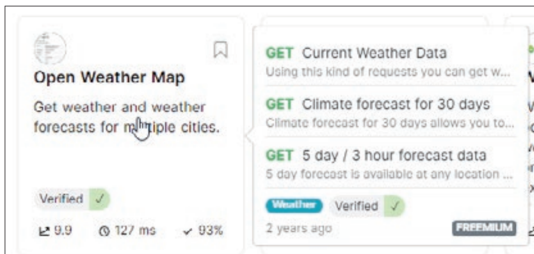
API (Application Programming Interface), czyli interfejs programowania aplikacji. Jest to przede wszystkim specyfikacja wytycznych, jak powinna przebiegać interakcja między komponentami programowymi. Implementacja API jest zestawem rutyn, protokołów i rozwiązań informatycznych do budowy aplikacji. Dodatkowo API może korzystać z komponentów graficznego interfejsu użytkownika. Dobre API ułatwia budowę oprogramowania, sprowadzając ją do łączenia przez programistę bloków elementów w ustalonej konwencji.
Definiuje się je na poziomie kodu źródłowego dla składników oprogramowania, na przykład aplikacji, bibliotek, systemu operacyjnego. Zadaniem interfejsu programowania aplikacji jest dostarczenie odpowiednich specyfikacji podprogramów, struktur danych, klas obiektów i wymaganych protokołów komunikacyjnych. Definicja API może być niezależna od jego implementacji. Co więcej, istnieją API zdefiniowane w sposób niezależny od danej platformy (systemu operacyjnego, języka programowania), które można wygenerować dla specyficznej platformy. Takie interfejsy definiuje się, używając zwykle języków ich opisu.
Jednym z typów API, które są szerzej znane, są API webowe (zwane też internetowymi).
Jest to rodzaj API, w których funkcje są udostępniane jako zasób w sieci. Bieżące wersje systemów API webowych pozwalają w bardzo łatwy sposób integrować informacje z sieci z aplikacjami, poszerzając ich funkcje lub umożliwiając współdziałanie (na przykład z sieciami społecznościowymi).
Właśnie z takiego API musimy skorzystać w naszym projekcie – ma to być API webowe, które pozwoli nam na integrację danych z naszą aplikacją.

Wybieramy odpowiednie API

W naszym projekcie potrzebujemy API z danymi pogodowymi. Musimy więc znaleźć dostawcę, który pozwoli nam wykorzystać dane pogodowe do naszej aplikacji. Na szczęście istnieje kilku różnych dostawców do tworzenia aplikacji pogodowych. Większość z nich oferuje bezpłatny pakiet wraz z subskrypcjami płatnymi, które różnią się ceną w zależności od usług i funkcji.
Na stronie **rapidapi.com/search/weather** znajdziemy pełny wybór API związanych z pogodą wraz z informacjami, które są płatne, a które nie są. Możemy również przeglądać pozostałe kategorie API – jest ich naprawdę wiele.



Zdecydujemy się na **OpenWeatherMap**, jest to jedno z najpopularniejszych darmowych API. OpenWeather opisuje siebie jako grupę ekspertów IT i naukowców zajmujących się danymi, którzy od 2014 roku zajmują się analizą danych pogodowych. Dla każdego punktu na Ziemi OpenWeather zapewnia wiarygodne historyczne, bieżące i prognozowane dane pogodowe za pośrednictwem interfejsów API.



Uzyskanie API key

W przypadku praktycznie każdego API konieczne jest uzyskanie specjalnego klucza, który potrzebny będzie do odebrania odpowiedzi z serwera zewnętrznego. W przypadku OpenWeatherMap API musimy dokonać rejestracji.

1 Przechodzimy do witryny o adresie **home.openweathermap.org/users/sign_up** i rejestrujemy się.

Guide API Dashboard Marketplace Pricing Maps Our Initiatives

Create New Account

...gmail.com

We will use information you provided for management and administration purposes, and for keeping you informed by mail,

2 Następnie w naszej skrzynce e-mail otwieramy wiadomość weryfikacyjną i klikamy na **Verify your email**.

OpenWeather

Dear Customer!

Thank you for choosing OpenWeatherMap!

Please confirm your email address to help us ensure your account is always protected.

Verify your email

3 Po chwili otrzymamy kolejną wiadomość, tym razem będzie zawierała ona nasz unikalny API key (klucz API) niezbędny do wysyłania zapytań do serwera.

OpenWeather

Dear Customer!

Thank you for subscribing to Free OpenWeatherMap!

API key:
- Your API key is 3942f2011
- Within the next couple of hours, it will be activated and ready to use
- You can later create more API keys on your account page
- Please, always use your API key in each API call

Endpoint:
- Please, use the endpoint api.openweathermap.org for your API calls
- Example of API call:
api.openweathermap.org/data/2.5/weather?q=London,uk&APPID=

Useful links:
- API documentation https://openweathermap.org/api
- Details of your plan https://openweathermap.org/price
- Please, note that 16-days daily forecast and History API are not available for Free subscribers

4 W ramach naszego darmowego klucza otrzymamy status **Free**. Daje on nam możliwość wykonywania 60 zapytań na minutę. W celach edukacyjnych jest to zupełnie wystarczające. W ramach tego konta mamy dostęp do

The Professional API collections contain access to weather historical data, current and various forecasts. Thes... request weather maps and get weather data not only via API, but with other instruments.

	Free	Startup	Developer	Profes...
Price is fixed, no other hidden costs (VAT is not included)	Get API key	30 GBP/month Subscribe	140 GBP/month Subscribe	370 GB... Subs...
API calls	60 calls/minute 1,000,000 calls/month	600 calls/minute 10,000,000 calls/month	3,000 calls/minute 100,000,000 calls/month	30,000 calls/... 1,000,000,00... calls/month

Price is fixed, no other hidden costs (VAT is not included)	Free	Startup 30 GBP/ month	Developer 140 GBP/ month	Professional 370 GBP/ month	Enterprise from 1500 GBP/ month
	Get API key	Subscribe	Subscribe	Subscribe	Subscribe
Current Weather API	✓	✓	✓	✓	✓
Hourly Forecast 4 days	-	-	✓	✓	✓
3-hour Forecast 5 days	✓	✓	✓	✓	✓
Daily Forecast 16 days	-	✓	✓	✓	✓
Climatic Forecast 30 days	-	-	✓	✓	✓
Bulk Download	-	-	-	✓	✓
Road Risk API (basic configuration)	-	-	-	-	✓
Air Pollution API	✓	✓	✓	✓	✓
Geocoding API	✓	✓	✓	✓	✓
Weather Triggers	✓	✓	✓	✓	✓

informacji o aktualnej pogodzie, jak również do 5-dniowej prognozy oraz kilku innych API.

Jak wysyłać zapytania do API

Jest to bardzo proste – wystarczy wejść na odpowiedni adres, który będzie zapytaniem typu GET.

Adresem, na który wysyłamy zapytanie, jest **api.openweathermap.org/data/2.5/weather**. Do adresu musimy jednak dołączyć parametry zapytania, czyli na przykład miejscowość i nasz API key oraz ewentualnie inne parametry.

W ten sposób uzyskamy dane pogodowe do naszej aplikacji webowej – musimy jednak wykonać zapytanie wewnątrz kodu, a następnie je przetworzyć, aby było czytelne dla użytkownika.

Grafika dla aplikacji

Po wybraniu API trzeba zastanowić się nad wyglądem naszej aplikacji. Warto poświęcić uwagę grafice – użytkownicy nie będą się zastanawiać, jak złożony jest kod od strony backendu, interesować ich będzie część wizualna i to, jak się korzysta z aplikacji. Bez graficznych dodatków nasza aplikacja będzie jedynie tekstową interpretacją danych uzyskanych z API.

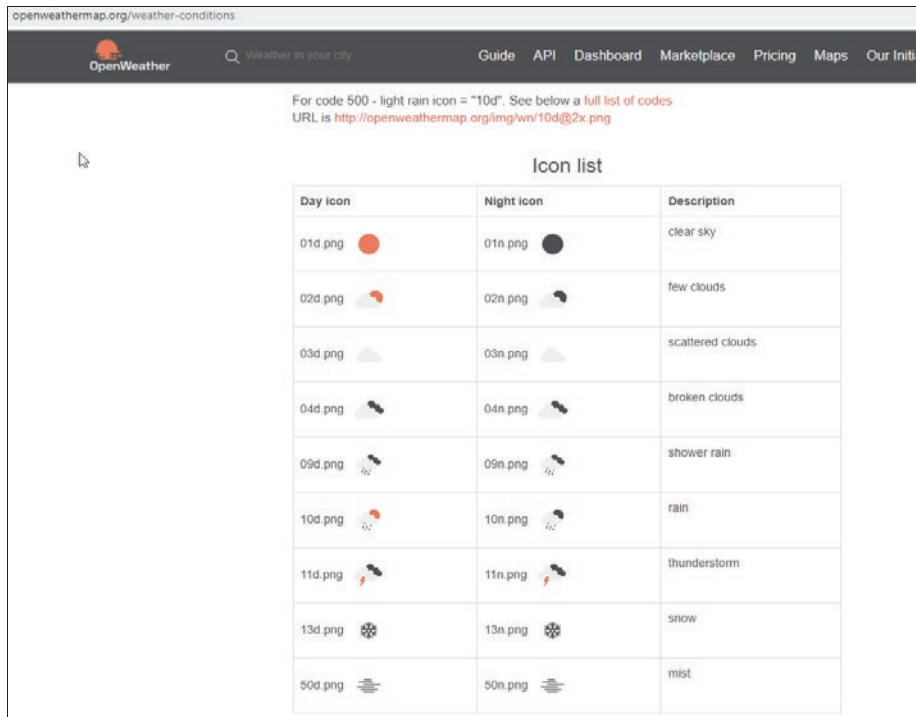
Nam będą potrzebne ikony informujące o pogodzie, symbolizujące deszcz, zachmurzenie, pełne słońce itp. Można stworzyć własne lub skorzystać z gotowych rozwiązań, na przykład kupując pakiet grafik. W naszym przypadku nie jest to konieczne – OpenWeatherMap za-



Przykład płatnego zestawu ikon pogodowych

api.openweathermap.org/data/2.5/weather?q=Warszawa&appid=3942f2[illegible]&units=metric&lang=pl

{"coord":{"lon":21.0118,"lat":52.2298},"weather":[{"id":800,"main":"Clear","description":"bezchmurnie","icon":"01d"}],"base":"stations
{"temp":27.09,"feels_like":27.09,"temp_min":25.78,"temp_max":28.28,"pressure":1015,"humidity":43},"visibility":10000,"wind":{"speed":4
{"type":2,"id":2032856,"country":"PL","sunrise":1656209757,"sunset":1656270083},"timezone":7200,"id":756135,"name":"Warszawa","cod":20



pewnia również darmowe ikony pogodowe, z których będziemy mogli skorzystać w naszej aplikacji.

Tworzymy plik HTML

1 Zaczynamy od stworzenia kodu HTML samej strony, musimy umieścić w nim odpowiednie miejsca na elementy aplikacji **A**.

2 Przede wszystkim warto zwrócić uwagę na wiersz 6 **B** – korzystamy tutaj z meta-tagu **viewport**. Odpowiada on za przechowywanie informacji na temat dostosowania naszej strony do urządzeń mobilnych. Nasza strona taka będzie, a parametry tego tagu informują, że strona zostanie rozciągnięta do 100 procent szerokości obszaru ekranu oraz pozostanie w skali 1:1 z dozwoloną możliwością skalowania.

3 W wierszu 8 **C** pokazana jest metoda wskazania zewnętrznego pliku **CSS**. W przypadku tej aplikacji będzie dużo różnych stylów. Gdybyśmy umieścili je w pliku HTML, stałby się on nieczytelny. Dzięki znacznikowi **link** możemy wskazać zewnętrzny plik ze stylami CSS.

4 Wewnątrz znacznika **body** umieszczamy dwa znaczniki **section**, które pozwolą na zdefiniowanie dwóch sekcji. Pierwsza sekcja **D** będzie zawierać nagłówek, formularz wyszukiwania i pusty element **span**. Ten element będzie widoczny z odpowiednim komunikatem pod pewnymi warunkami. W szczególności, jeśli nie są dostępne żadne dane pogodowe dla żądanego miasta lub dane dla tego miasta są już znane. Dodatkowo do pola wprowadzania dodajemy atrybut **autofocus**. Dzięki temu po załadowaniu strony znacznik pisania będzie ustawiony w tym polu.

5 Druga sekcja **E** jest przeznaczona na listę miast. Domyślnie nie będzie w niej miast, ale gdy zaczniemy szukać pogody dla konkretnego miasta, jeśli dane pogodowe są dostępne, odpowiedni element (miasto) zostanie dołączony do listy. Dodatkowo poza znacznikiem **sec-**

Help

Przyklad.html • pogoda.html • pogoda-style.css

```
<!DOCTYPE html>
<html lang="pl-PL">
<head>
    <meta charset="UTF-8">
    <meta http-equiv="X-UA-Compatible" content="IE=edge">
    <meta name="viewport" content="width=device-width, initial-scale=1.0">
    <title>Aplikacja pogodowa</title>
    <link rel="stylesheet" href="pogoda-style.css">
</head>
<body>
      <section class="top-banner">
        <div class="container">
          <h1 class="heading">Aplikacja pogodowa</h1>
          <form>
            <input type="text" placeholder="Podaj nazwę miejscowości" autofocus>
            <button type="submit">WYŚLIJ</button>
            <span class="msg"></span>
          </form>
        </div>
      </section>
      <section class="ajax-section">
        <div class="container">
          <ul class="cities"></ul>
        </div>
      </section>
      <script src="pogoda.js"></script>
</body>
</html>
```

tion umieszczamy znacznik **script** i wskazujemy nazwę pliku z kodem JavaScript.

Określamy style CSS

Nasz plik będzie zawierał style niezbędne do prezentacji tła, resetowania i obsługi różnych elementów aplikacji **A** (następna strona).

To co kluczowe, a wcześniej nie było omawiane, to prezentacja naszej aplikacji przy różnych rozdzielczościach.

Ponieważ dostosowujemy naszą aplikację do ekranów mobilnych i będziemy wyświetlać w niej potencjalnie wiele różnych miast w trakcie jednej sesji, musimy odpowiednio zadbać o prezentację w zależności od rozdzielczości ekranu użytkownika. Skorzystamy do tego z opcji **@media screen and (max-width: 700px)**

Wiersze 203-222 **B** pozwalają na zdefiniowanie prezentacji sekcji pierwszej na dwa sposoby. Jeśli wartość szerokości dla rozdzielczości okna przeglądarki jest mniejsza niż 700px, obszar formularza zostanie podzielony na dwa wiersze. W przypadku okna o wyższej rozdzielczości informacja będzie prezentowana w jednym wierszu.

W przypadku sekcji drugiej, w której będą informacje pogodowe, w zależności od rozdzielczości będziemy prezentować cztery, trzy, dwie lub jedną kolumnę z danymi. Ma to duże znaczenie w przypadku smartfonów – nasza aplikacja jest bardzo elastyczna i prezentuje się dobrze w dowolnej rozdzielczości.

Dodajemy kod JavaScript

Po utworzeniu tych elementów jesteśmy gotowi na tworzenie samego kodu JS. Najczęściej tworzy się te trzy elementy jednocześnie, dodając lub odejmując fragmenty kodu, tak aby uzyskać satysfakcjonujący efekt. W każdej chwili może się okazać, że czegoś nie przewidzieliśmy, że na przykład trzeba dodać przycisk w głównym widoku, który musimy zakodować w kodzie HTML, sformatować nowym stylem

A

```css
/* STYLE DO RESETU
                                                  */
:root {
    --bg_main: #0a4144;
    --text_light: #fff;
    --text_med: #53627c;
    --text_dark: #1e2432;
    --red: #ff1e42;
    --darkred: #c3112d;
    --orange: #ff8c00;
  }

  a {
    color: inherit;
    text-decoration: none;
  }

  * {
    margin: 0;
    padding: 0;
    box-sizing: border-box;
    font-weight: normal;
  }

  button {
    cursor: pointer;
  }

  input {
    -webkit-appearance: none;
  }

  button,
  input {
    border: none;
    background: none;
```

No results found for 'const filteredArray'

B

```css
@media screen and (max-width: 700px) {
    .heading,
    .ajax-section .city-temp {
      font-size: 3rem;
    }

    .ajax-section {
      margin-top: 20px;
    }

    .top-banner form {
      flex-direction: column;
      align-items: flex-start;
    }

    .top-banner form input,
    .top-banner form button {
      width: 100%;
    }

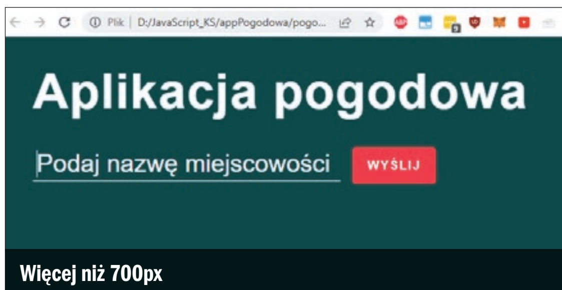
    .top-banner form button {
      margin: 20px 0 0 0;
    }

    .top-banner form .msg {
      position: static;
      max-width: none;
```

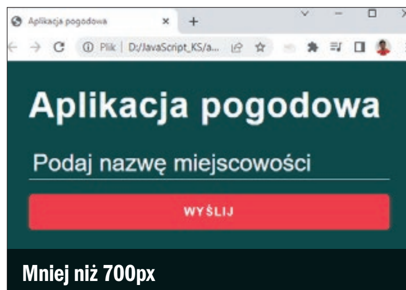
w pliku CSS, a w kodzie JS musimy przypisać mu instrukcje.
W naszym kodzie chcemy umieścić instrukcje, które sprawią, że za każdym razem, gdy użytkownik w pole tekstowe wprowadzi nazwę miasta i kliknie na przycisk **WYŚLIJ**, pojawi się informacja o aktualnej pogodzie w danym mieście.

1 Na początku deklarujemy zmienne, z których będziemy korzystali A.

2 Po kliknięciu przez użytkownika na **WYŚLIJ** musimy zatrzymać przesyłanie formularza, aby zapobiec ponownemu załadowaniu strony oraz przechwycić wartość z pola tekstowego B.



Więcej niż 700px



Mniej niż 700px

```
@media screen and (max-width: 500px) {
  body {
    padding: 15px;
  }

  .ajax-section .cities {
    grid-template-columns: repeat(1, 1fr);
  }
}
```

Kod stylu dla układu jednokolumnowego



Prezentacja pełnego układu czterokolumnowego dla dużych rozdzielczości



Prezentacja układu dwukolumnowego dla mniejszych ekranów, w tym urządzeń mobilnych

3 Musimy wykonać żądanie **AJAX**, aby uzyskać dane pogodowe dla naszej aplikacji. (Tu warto wyjaśnić, co oznacza ten skrót – patrz ramka na kolejnej stronie). Przy pierwszym wysłaniu żądania przez użytkownika zakładamy, że nasza aplikacja nie wykonywała żadnego żądania AJAX i dopiero na tym etapie wykonamy żądanie do OpenWeatherMap API i przekażemy w żądaniu parametry:

- nazwa miasta, która będzie wartością pobraną z pola wyszukiwania,
- klucz API, nasz prywatny klucz uzyskany podczas procesu rejestracji w OpenWeatherMap API, konieczny do uzyskania danych z serwera,
- jednostki temperatury – w naszym przykładzie stopnie Celsjusza,
- język danych, domyślnie dane otrzymywane z serwera są w języku angielskim, my oczekujemy danych w języku polskim.

A

```
const form = document.querySelector(".top-banner form");
const input = document.querySelector(".top-banner input");
const msg = document.querySelector(".top-banner .msg");
const list = document.querySelector(".ajax-section .cities");

var apiKey = "WPROWADZ SWOJ KLUCZ TUTAJ";
```

B

```
form.addEventListener("submit", e => {
  e.preventDefault();
  const listItems = list.querySelectorAll(".ajax-section .city");
  const inputVal = input.value;
```

4 To, co chcemy uzyskać, musimy zweryfikować z dokumentacją OpenWeatherMap API, czy i jakimi parametrami możemy otrzymać oczekiwane przez nas rezultaty. Po dokładnej analizie dokumentacji uzyskamy odpowiednią wartość zapytania.

5 Zgodnie z dokumentacją w zapytaniu podajemy

AJAX

AJAX (Asynchronous JavaScript and XML) to technologia pozwalająca na odświeżanie jedynie części strony, a nie jej całości, co odbywa się asynchronicznie. Dane są wówczas zaciągane z pliku XML lub TXT zlokalizowanego na serwerze za pomocą JavaScriptu albo innego skryptu.

AJAX nie jest zatem językiem programowania, lecz techniką wykorzystującą inne języki. Znajduje zastosowanie przy tworzeniu interaktywnych stron, na których dane przetwarza się bez konieczności przeładowania całej strony. Wykorzystuje się ją też przy pobieraniu danych z zewnętrznego API, danych o produktach dostępnych w sklepie internetowym czy przedłużaniu w tle sesji aktywnego użytkownika.

Dzięki asynchroniczności działanie przeglądarki nie zostaje zablokowane przez skrypt, a przesyłanie danych traktuje się jako odrębny proces realizowany w tle. W ten sposób przeglądarka działa znacznie szybciej, niż miałoby to miejsce w przypadku działania synchronicznego.

Co się składa na AJAX

Podstawowe elementy wchodzące w skład techniki AJAX to:

- **XML** – jest językiem znaczników, za pomocą którego opisuje się odbierane dane. Często okazuje się jednak, że informacje przesyłane są w innym formacie i odbierane w postaci tekstowej. Mowa tu o gotowych fragmentach HTML czy kodu JavaScript.
- **JavaScript** – nie stanowi warunku koniecznego, ponieważ zamiast niego można użyć innego języka skryptowego, na przykład VBScript bądź JScript.
- **XMLHttpRequest** – (XHR) pozwala na asynchroniczne wysyłanie danych. W ten sposób dane mogą być pobierane w tym samym czasie z różnych miejsc, a użytkownik może wykonywać w trakcie przesyłania danych inne czynności.

W czystym JavaScripcie do obsługi dynamicznych połączeń (XHR) możemy skorzystać z dwóch obiektów, czyli **XMLHttpRequest** oraz **Fetch**.

Ten pierwszy istnieje od początku istnienia AJAX. Mnogość opcji i to, że trzeba napisać bardzo dużo kodu, sprawia, że w realnych projektach rzadko się korzysta z tej opcji.

Ten drugi w wielu sytuacjach będzie prostszy w użyciu, a to dzięki domyślnej obsłudze **Promise** (obietnic) oraz kilku nowym elementom, takim jak **Headers**, **Response**, **Request**. Trzeba mieć jednak na uwadze, że nie każda przeglądarka go obsłuży.

odczytane z żądania użytkownika miasto, przesyłamy nasz klucz API, następnie dzięki parametrowi **units=metric** C uzyskamy dane w systemie metrycznym, a parametr **lang=pl** D – pozwala na uzyskanie danych w języku polskim.

Fetch – jak wykonać żądanie AJAX

Jak już wiemy, aby wykonać żądanie AJAX, mamy wiele opcji. Możemy użyć zwykłego XMLHttpRequest, nowszego Fetch, a nawet biblioteki JavaScript, takiej jak jQuery i Axios. W tym przykładzie użyjemy **Fetch**. Aby pobrać żądane dane, musimy wykonać następujące czynności:

1. Trzeba przekazać adres URL, do którego chcemy uzyskać dostęp do metody **fetch()**.

2. **Fetch()** zwróci **Promise** zawierającą odpowiedź (obiekt **Response**). Nie będzie to jeszcze rzeczywista odpowiedź,

C D

```
const url = `https://api.openweathermap.org/data/2.5/weather?q=${inputVal}&appid=${apiKey}&units=metric&lang=pl`;
```

tylko odpowiedź HTTP. Aby pobrać dane odpowiedzi w żądanym formacie JSON (jest to domyślny format danych OpenWeatherMap), użyjemy metody **json()** obiektu Response.

```
fetch(url)
  .then(response => response.json())
  .then(data => {
    // TUTAJ ZNAJDZIE SIĘ RESZTA KODU
  })
  .catch(() => {
    msg.textContent = "Nie znalezionego szukanego miasta 😩"; A
  });
```

3 Ta metoda zwraca kolejną Promise. Gdy zostanie spełniona, dane będą dostępne do manipulacji.

4 Jeśli z jakiegoś powodu żądanie nie powiedzie się, na ekranie pojawi się odpowiedni komunikat **A**.

Weryfikacja zapytania

Możemy już teraz zweryfikować działanie zapytania i uzyskać dostęp do danych odpowiedzi.

1 Po uruchomieniu aplikacji w przeglądarce od razu uruchamiamy konsolę, a następnie przechodzimy do zakładki **Network** **A**.

2 Teraz podajemy nazwę miasta do pola wyszukiwania i klikamy na **WYŚLIJ**. Na liście zapytań pojawi się odpowiedź uzyskana z serwera – klikamy na nią **B**.

3 Przechodzimy do zakładki **Preview** **C**. W tym widoku będziemy mogli podejrzeć dane w formacie JSON, które musimy obsłużyć w naszej aplikacji. Ilość danych jest ogromna, wykorzystamy tylko niektóre z nich.

× Headers Payload Preview Response Ini **C**

▾{coord: {lon: 21.0118, lat: 52.2298},…}
base: "stations"
▸clouds: {all: 0}
cod: 200
▸coord: {lon: 21.0118, lat: 52.2298}
dt: 1656241774
id: 756135
▸main: {temp: 29.69, feels_like: 29.04, te
name: "Warszawa"
▸sys: {type: 2, id: 2032856, country: "PL"
timezone: 7200
visibility: 10000
▸weather: [{id: 800, main: "Clear", descri
▸wind: {speed: 7.2, deg: 140}

4 Jeśli po wpisaniu poprawnej nazwy miasta otrzymamy błąd „Nie znaleziono szukanego miasta", otwieramy konsolę przeglądarki. Możemy tam znaleźć informacje o błędach. W tym przypadku otrzymaliśmy z OpenWeatherMap API odpowiedź z kodem 401, co oznacza, że nasze zapytanie nie przeszło walidacji

Aplikacja pogodowa

D:/JavaScript_KS/appPogodowa/pogoda.html

Aplikacja pogodowa

Podaj nazwę miejscowośc

WYŚLIJ

Elements Console Recorder Performance insights Sources Network **A** Performance

Name: pogoda.html, pogoda-style.css, pogoda.js, weather?q=Warszawa&appid...

Headers Preview Response Initiator Timing

▾General
Request URL: file:///D:/JavaScript_KS/appPogodowa/pogoda.html
Request Method: GET
Status Code: 200 OK
Referrer Policy: strict-origin-when-cross-origin

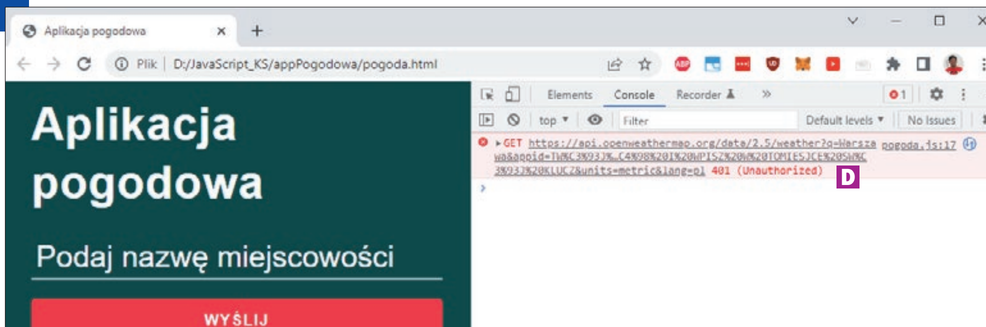
▾Response Headers
Content-Type: text/html
Last-Modified: Sun, 26 Jun 2022 09:04:32 GMT

Name: pogoda.html, pogoda-style.css, pogoda.js, weather?q=Warszawa&appid... **B**

Headers Payload Preview Response Initiator Timing

▾General
Request URL: https://api.openweathermap.org/data/2.5/weather?q=Warszawa&appid=[redacted]&units=metric&lang=pl
Request Method: GET
Status Code: 200 OK
Remote Address: 37.139.1.159:443
Referrer Policy: strict-origin-when-cross-origin

▾Response Headers View source
Access-Control-Allow-Credentials: true
Access-Control-Allow-Methods: GET, POST



i należy podać odpowiednie dane autoryzujące **D**. W takim przypadku należy zweryfikować, czy mamy podany poprawny API key w kodzie JS.

Zebranie potrzebnych danych

Po uruchomieniu żądania AJAX za każdym razem, gdy wpiszemy miasto w polu wyszukiwania, API zwróci dane pogodowe, jeśli są dostępne. Naszym zadaniem jest teraz zebranie tylko tych danych, których potrzebujemy, a następnie utworzenie powiązanego elementu listy i na koniec dodanie go do tej listy.

```
    const li = document.createElement("li");
    li.classList.add("city");
    const markup = `
      <h2 class="city name" data-name="${name},${sys.country}">
        <span>${name}</span>
        <sup>${sys.country}</sup>
      </h2>
      <div class="city-temp">${Math.round(main.temp)}<sup>°C</sup></div
      <figure>
        <img class="city-icon" src=${icon} alt=${weather[0]["main"]}>
        <figcaption>${weather[0]["description"]}</figcaption>
      </figure>
    `;
    li.innerHTML = markup;
    list.appendChild(li);
  })
```

1 W wierszu 20 deklarujemy zmienne, które będą zawierać odpowiedź z serwera, w wierszu 21–23 deklarujemy zmienną **icon**, która pozwoli na wyświetlanie ikon pasujących do pogody po kodzie zwracanym z odpowiedzi zapytania API. Jest to zgodne z dokumentacją OpenWeatherMap API.

```
20    const { main, name, sys, weather } = data;
21    const icon = `https://openweathermap.org/img/wn/${
22      weather[0]["icon"]
23    }@2x.png`;
```

2 Następnie tworzymy listę i jej nowy element, do którego przypisujemy poszczególne wartości uzyskane z zapytania API. Zaokrąglamy też odczyt temperatury. OpenWeatherMap API dostarcza danych dotyczących temperatury z dokładnością do dwóch miejsc po przecinku. W naszej aplikacji chcemy wartości całkowitych, dlatego korzystamy z metody **math.round()** **A**.

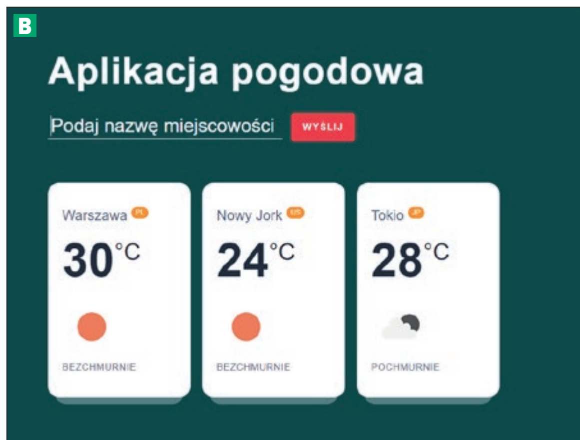
3 Na koniec po poprawnym obsłużeniu żądania AJAX musimy wyczyścić wszystkie niezbędne elementy, aby przygotować się na kolejne żądanie użytkownika. Czyścimy zawartość zmiennej **msg**, czyścimy formularz oraz umieszczamy znacznik wpisywania w pole wprowadzania. Gotowe – teraz możemy przystąpić do korzystania z własnej aplikacji pogodowej **B**.

```
    msg.textContent = "";
    form.reset();
    input.focus();
  });
```

Jak rozbudować aplikację

Możemy rozszerzyć wyświetlane informacje. Wystarczy dodać kilka linii kodu, aby wyświetlić kolejne dane, które otrzymujemy z API.

1 Poniżej znacznika **<figure>** **C** dodajemy kolejny, ale z parametrem dotyczącym wartości ciśnienia, którą uzyskamy z wartości **main.pressure**.



2 Po zapisaniu pliku, odświeżeniu aplikacji i wyszukaniu konkretnego miasta poznamy dane dotyczące ciśnienia **D**.



3 Jeśli chcemy dodać informacje o prędkości wiatru, będziemy musieli wprowadzić nieco więcej zmian. W sekcji odpowiedzialnej za przypisywanie do zmiennych wartości z zapytania musimy dodać wpis **wind**.

```
        const { main, name, sys, weather, wind } = data;
```

4 Następnie trzeba dodać kolejny element, **figure**, w którym musimy obsłużyć zmienną odpowiadającą predkości wiatru.

```
        <figure>
          <figcaption> Wiatr: ${Math.round(wind.speed*3.6)} km/h</figcaption>
        </figure>
```

E

5 W odpowiedzi z serwera zgodnie z dokumentacją API otrzymujemy informacje o prędkości wiatru wyrażoną w metrach na sekundę. Aby te dane były bardziej czytelne, musimy przekonwertować otrzymaną wartość na jednostkę km/h. Robimy to poprzez mnożenie otrzymanej wartości razy 3.6 **E** (w kodzie stosujemy zapis angielski). Następnie wykonujemy zaokrąglenie do liczb całkowitych.



6 Po zapisaniu pliku, odświeżeniu aplikacji i wyszukaniu miasta poznamy również dane dotyczące wiatru **F**.

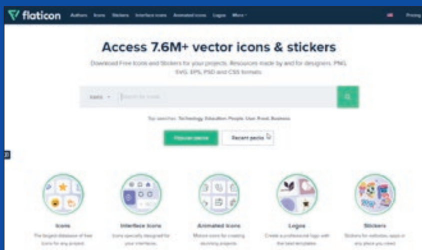
C

```
        </figure>
        <figure>
          <figcaption>Ciśnienie: ${main.pressure} hPa</figcaption>
        </figure>
      `;
```

DODAJEMY IKONY DLA WIATRU I CIŚNIENIA

Warto dodać ikony dla nowo wyświetlanych informacji dotyczących ciśnienia i wiatru. Tylko że takich ikon nie znajdziemy w bazie OpenWeatherMap API, musimy je wyszukać sami.

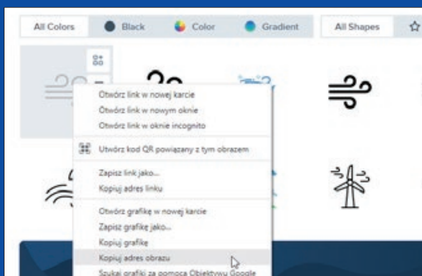
1 Wchodzimy na witrynę **flaticon.com**, na której możemy znaleźć tysiące darmowych ikon.



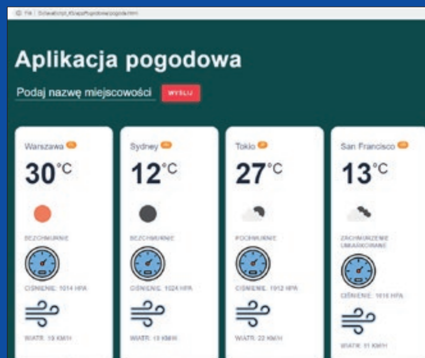
2 Wpisujemy na przykład frazę **wind** i wyszukujemy.



3 Następnie klikamy prawym przyciskiem myszy na ikonę, która nam odpowiada, i wybieramy opcję **Kopiuj adres obrazu**.



4 Teraz w kodzie JS dodajemy w obszarze **figure** obiekt **img** i wskazujemy jako adres skopiowany adres wybranej ikony **A**. Po zapisaniu pliku, odświeżeniu aplikacji i wyszukaniu konkretnego miasta zobaczymy nowe ikony w oknie aplikacji.



Uwaga! Korzystając z ikon umieszczanych na serwisach zewnętrznych, powinniśmy dodatkowo umieścić skrypt do weryfikacji linków, tak abyśmy byli informowani, gdy na naszej stronie przestaną funkcjonować odnośniki, które wskazują dokładny adres pliku z ikoną.
Można oczywiście dostosować rozmiar ikony oraz jej położenie w każdej z sekcji

```
    .ajax-section .city-icon-wind {
      margin-top: 10px;
      margin-left: 30px;
      width: 50px;
      height: 50px;
    }

    .ajax-section .city-icon-pressure {
      margin-top: 10px;
      margin-left: 25px;
      width: 50px;
      height: 50px;
    }
```

```
          <img class="city-icon" src='https://cdn-icons-png.flaticon.com/128/5132/5132956.png'>
          <figcaption>Ciśnienie: ${main.pressure} hPa</figcaption>
        </figure>
        <figure>
          <img class="city-icon" src='https://cdn-icons-png.flaticon.com/512/615/615775.png'>
          <figcaption> Wiatr: ${Math.round(wind.speed*3.6)} km/h</figcaption>
        </figure>
      `;
```

A

```
      <img class="city icon" src=${icon} alt=${weather[0]["main"]}>
      <figcaption>${weather[0]["description"]}</figcaption>
    </figure>
    <figure>
      <img class="city icon pressure" src='https://cdn-icons-png.flaticon.com/128/5132/5132956.png'>
      <figcaption>Ciśnienie: ${main.pressure} hPa</figcaption>
    </figure>
    <figure>
      <img class="city icon wind" src='https://cdn-icons-png.flaticon.com/512/615/615775.png'>
      <figcaption> Wiatr: ${Math.round(wind.speed*3.6)} km/h</figcaption>
    </figure>
  `;
  li.innerHTML = markup;
```

– wystarczy dodać odpowiedni opis stylu w pliku CSS w sekcji drugiej. Oraz zmienić nazwę **class** w ramach, w jakich mają być formatowane nowe ikony w pliku z kodem JS. Zmiany można obejrzeć po zapisaniu dokumentów i ponownym załadowaniu strony.



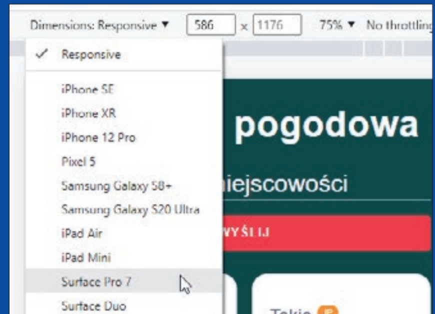
TESTUJEMY APLIKACJĘ W TRYBIE MOBILNYM

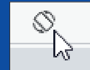
Jak już wiemy, nasza aplikacja powinna odpowiednio się skalować i zmieniać swój wygląd w zależności od rozdzielczości. Google Chrome umożliwia przetestowanie na komputerze działania aplikacji czy witryny w trybie mobilnym, symulując jej wyświetlanie tak, jakby miało miejsce na ekranie smartfona.

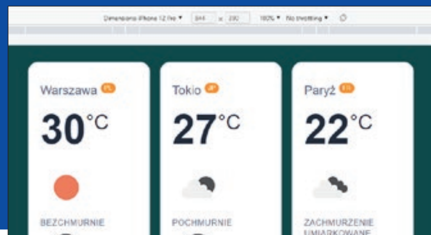
1 Uruchamiamy naszą aplikację w przeglądarce Google Chrome, a następnie korzystamy ze skrótu klawiaturowego [ctrl]+[shift]+[M] lub klikamy na symbol smartfona na pasku narzędzi deweloperskich.

2 W polu **Dimensions** wybieramy jedno z urządzeń lub wskazujemy rozdzielczość.

3 Następnie możemy przetestować działanie aplikacji. W celu wykonania obrotu ekranu o 90 stopni klikamy na górnym pasku na ikonę obrotu.







7 HTML, CSS i JavaScript w pigułce

Ten rozdział to nie tylko podsumowanie wiadomości o HTML, CSS i JS. Poznamy w nim też znaczniki, które do tej pory się nie pojawiły, a są przydatne. W rozdziale tym są omówione najważniejszych elementy języka JavaScript

Podstawowe znaczniki, z jakich zbudowane są dokumenty HTML, to **html**, **head** i **body**. Wiemy już, że pierwszy z nich otacza całość dokumentu. Drugi odpowiada za funkcjonowanie strony, a trzeci za jej treść. Przyjrzyjmy się innym istotnym znacznikom.

```
<!DOCTYPE html>
<html>
    <head>

    </head>
    <body>

    </body
</html>
```

Szkielet dokumentu, od którego zazwyczaj rozpoczynaliśmy pracę z HTML-em

Podstawowe znaczniki HTML

Do tej pory omówiliśmy jedynie kilka znaczników wpływających na wygląd strony. Zwróćmy uwagę na to, że układ wielu stron internetowych jest podobny. Jest tak dlatego, że w języku HTML5 znajdują się znaczniki blokowe, dzielące stronę na logiczne części.

ARTICLE

Jednym z takich znaczników specyficznych dla HTML5 jest **article**, czyli artykuł.
To znacznik, którym wydziela się samodzielną część strony.
Dobrze sprawdza się na przykład w przypadku bloga czy innych tekstów, pomaga w sytuacji, gdy po dodaniu obrazów do wpisów treść zaczyna się nieco rozjeżdżać, a także wtedy, gdy chcemy po prostu uporządkować teksty.
Oto przykładowy efekt:



Znacznik **article** może pomóc w naprawie tak wyglądającej strony. Można wykorzystać go do wydzielenia poszczególnych wpisów.

```
<article class="browser">
  <h2>Google Chrome</h2>
  <p>Pierwsza wersja beta została wydana 2
    Według serwisu Statcounter.com w stycz
</article>
<article class="browser">
  <h2>Mozilla Firefox</h2>
  <p> wieloplatformowa przeglądarka interne
     Mozilla Corporation oraz wolontariuszy
</article>
<article class="browser">
```

Na wpis składa się nagłówek i paragraf. Te dwa elementy powinny być otoczone znacznikiem **article** – dla każdej takiej pary. Samo dodanie znacznika nie poprawi jeszcze wyglądu strony, ale bardzo ułatwi nam to zadanie. Będziemy mogli zdefiniować wygląd dla każdego znacznika **article**. Jeśli poprzez CSS nadamy temu znacznikowi określoną wysokość (**height**), to elementy z różnych wpisów nie będą już na siebie nachodzić.

```
article
{
    height: 220px;
}
```

SECTION

Zwykle poszczególne artykuły, czyli znaczniki **article**, tworzą pewną spójną całość. W przypadku naszego pierwszego projektu, bloga, artykuły możemy połączyć w sekcję poprzez otoczenie ich znacznikiem **section**. W taki sposób możemy wydzielić sekcję główną strony, obok której możemy dać sekcję boczną zawierającą treści mniej istotne dla samej strony.

```
<section>
<article class=
  <h2>Mozilla F
  <p> wieloplat
     Mozilla Co
</article>
```

ASIDE

Znacznik **aside** reprezentuje sekcję poboczną. Może ona być uważana za odrębny element strony, zwykle służy do prezentowania specjalnych, odrębnych treści, na przykład reklam, notki o autorze lub informacji o portalu, na którym znajduje się strona, czy też

o innych promowanych, powiązanych stronach. Znacznika tego używa się także często do dodawania ramek wyjaśniających pojawiające się w tekstach terminy.

MAIN

Wpisy na blogu oznaczone jako jedna wspólna sekcja i sekcja boczna **aside** mogą być ujęte razem jako część główna strony, czyli najważniejszy zasób treści. Do takiego oznaczenia używa się znacznika **main**.

HEADER

Jeśli mówimy o treści głównej, to należy wspomnieć jeszcze o tym, co jest wspólne dla całej strony, czyli o nagłówku. Do nagłówków tekstowych służą znaczniki **h1** (oraz **h2**, **h3**, **h4**, **h5** aż do **h6** – oznaczające kolejne, coraz niższe stopnie nagłówków). Nagłówkiem całej strony jest znacznik **header**. W nim można umieścić na przykład logo i inne elementy, które chcemy mieć oznaczone w kodzie strony jako jej część nagłówkowa.

FOOTER

Oprócz znacznika odpowiedzialnego za część nagłówkową strony jest też znacznik dla stopki strony. To znacznik **footer** – umieszcza się go na końcu i zapisuje w nim na przykład informację o autorze.

```
    <footer> 2022, Krzysztof Dziedzic</footer>
    </body>
</html>
```

Selektory

Przyjrzyjmy się ogólnym możliwościom CSS. Podstawą jest to, że możemy konkretnym elementom strony nadawać odpowiedni wygląd.
Zacznijmy od określenia elementów, jakim nadajemy wygląd. I tu pojawia się słowo **selektor** – jest to coś, co pozwala wyselekcjonować element z dokumentu HTML, tak by można było „osobno” zdefiniować dla niego wygląd. Ogólny zapis wyglądu w CSS ma następującą formę:

```
selektor
{
    właściwość_1: wartość;
    właściwość_2: wartość;
    ...
}
```

W miejsce słowa **selektor** możemy podstawiać różne elementy. Jakie?

1 W zadaniach, które zrealizowaliśmy w poprzednich rozdziałach, selektorem, jakiego najczęściej używaliśmy, była nazwa znacznika. W ten sposób odnosiliśmy się do

```
nazwa_znacznika
{
    właściwość_1: wartość;
    właściwość_2: wartość;
    ...
}
```

wszystkich znaczników o podanej nazwie, jakie znalazły się w dokumencie.

2 Innym omawianym przez nas selektorem były klasy. Jeśli chcemy nadać styl znacznikom konkretnego typu, które wcześniej umieściliśmy w jednej klasie, jako selektor podajemy najpierw nazwę znacznika, a potem nazwę klasy (po kropce).
Oto schemat:

```
nazwa_znacznika.nazwa_klasy
{
    właściwość_1: wartość;
    właściwość_2: wartość;
    ...
}
```

3 Jeśli mamy klasę, do której przypisaliśmy wiele znaczników różnych typów, i dla nich wszystkich chcemy nadać wspólny styl, jako selektor podajemy nazwę klasy poprzedzoną kropką. Oto schemat:

```
.nazwa_klasy
{
    właściwość_1: wartość;
    właściwość_2: wartość;
    ...
}
```

4 W naszych przykładach pojawiło się także nadawanie stylu znacznikom zagnieżdżonym. Jeśli chcemy nadać styl znacznikom danego rodzaju, ale tylko tym, które wcześniej umieszczono w innym konkretnym znaczniku, jako selektor najpierw podajemy nazwę znacznika zewnętrznego, wewnątrz którego znajdują się znaczniki, którym chcemy nadać styl, a po spacji nazwę znacznika, dla którego styl nadajemy. Oto schemat:

```
znacznik_zewnętrzny znacznik_wewnętrzny
{
    właściwość_1: wartość;
    właściwość_2: wartość;
    ...
}
```

5 Innym sposobem nadawania stylu wybranym znacznikom jest selektor wskazujący na **id** znacznika. By nadać styl znacznikowi o konkretnym **id**, jako selektor podajemy właśnie **id**, ale poprzedzamy je znakiem **#**. Oto schemat:

```
#id_znacznika
{
    właściwość_1: wartość;
    właściwość_2: wartość;
    ...
}
```

6 Kolejny sposób podawania selektora to połączenie dwóch nazw plusem. Wbrew pozorom nie zadziała to tak, że nada dwóm grupom znaczników taki sam wygląd. Ta metoda opisuje styl tylko dla znacznika podanego po plusie i to tylko wtedy, gdy występuje bezpośrednio po znaczniku podanym przed plusem. I tak **img + h1** nada styl znacznikowi **h1**, jeśli ten występuje bezpośrednio po znaczniku **img**. Oto schemat:

```
znacznik_poprzedzający + nazwa_znacznika
{
    właściwość_1: wartość;
    właściwość_2: wartość;
    ...
}
```

7 Nieco podobne jest łączenie znaczników poprzez znak **>**. Ten sposób również pozwala na nadanie stylu znacznikowi podanemu po znaku, jednak odnosi się do znaczników zagnieżdżonych i nadaje im styl tylko wtedy, gdy są bezpośrednimi „dziećmi” znacznika podanego przed znakiem. Nie mogą być zagnieżdżone w „dziecku” znacznika przed znakiem. Oto schemat:

```
nazwa_rodzica > nazwa_znacznika
{
    właściwość_1: wartość;
    właściwość_2: wartość;
    ...
}
```

8 Jeśli chcemy nadać styl pierwszemu znacznikowi zagnieżdżonemu w podanym znaczniku (jego pierwsze „dziecko”), możemy po nazwie znacznika dać znak dwukropka i napisać **first-child**. Oto schemat:

```
nazwa_rodzica:first-child
{
    właściwość_1: wartość;
    właściwość_2: wartość;
    ...
}
```

9 Analogicznie jak do pierwszego „dziecka” możemy odnieść się też do ostatniego znacznika zagnieżdżonego w znaczniku, jednak w przeciwieństwie do poprzedniego przykładu piszemy w tym wypadku **last-child** zamiast first-child. Oto schemat:

```
nazwa_rodzica:last-child
{
    właściwość_1: wartość;
    właściwość_2: wartość;
    ...
}
```

10 Do dwóch wcześniej opisanych przykładów podobne jest wykorzystanie **only-child**, które pozwala na nadanie stylu znacznikowi, jeśli jest on jedynym „dzieckiem” podanego przed dwukropkiem znacznika. Oto schemat:

```
nazwa_rodzica:only-child
{
    właściwość_1: wartość;
    właściwość_2: wartość;
    ...
}
```

```
nazwa_znacznika_występującego_wcześniej ~ nazwa_znacznika
{
    właściwość_1: wartość;
    właściwość_2: wartość;
    ...
}
```
A

11 Kolejny znak, którym łączymy znaczniki w selektorach, to **~**. To łączenie jest podobne do łączenia poprzez plus. Nie odnosi się jednak tylko do pierwszego znacznika po znaczniku przed znakiem, ale do wszystkich znaczników podanych po znaku, występujących po znaczniku podanym przed znakiem. Oto schemat **A**.

12 Niektórym znacznikom nadawane są różne wartości atrybutów. Przykładem może być **canvas** ze zrealizowanej przez nas gry platformowej, który miał nadane **id**, **width** i **height**. Samo nadanie znacznikowi wartości dla konkretnego atrybutu także może posłużyć jako selektor. I tak w wypadku znacznika canvas zapis **canvas[height]** wskazuje nie tylko na jeden canvas, ale też na każdy inny, w którym został użyty atrybut height. Uogólniając, tworząc selektor oparty na atrybucie, najpierw podajemy nazwę znacznika, a następnie w nawiasie kwadratowym nazwę atrybutu. Oto schemat:

```
<canvas id="gra" width="1280px" height="580px"> </canvas>
```

```
nazwa_znacznika[nazwa_atrybutu]
{
    właściwość_1: wartość;
    właściwość_2: wartość;
    ...
}
```

13 Nieco podobnym do poprzedniego, jednak jeszcze „węższym" selektorem jest możliwość wybrania znacznika z atrybutem o konkretnej wartości. Wtedy – w odniesieniu do poprzedniego przykładu – po nazwie atrybutu podajemy znak równości i w cudzysłowie wartość, jaką musi mieć ten atrybut, abyśmy wyselekcjonowali znacznik.
Oto schemat:

```
nazwa_znacznika[nazwa_atrybutu="wartość_atrybutu"]
{
    właściwość_1: wartość;
    właściwość_2: wartość;
    ...
}
```

14 Tworząc selektory, możemy też negować. Załóżmy, że chcemy nadać styl wszystkim znacznikom **div**, ale z wyłączeniem wybranych, na przykład bez znacznika **div** i **id pole**. Wtedy zapiszemy to w ten sposób **div:not(#pole)**. Oto schemat:

```
nazwa_znacznika:not(co_wykluczyć)
{
    właściwość_1: wartość;
    właściwość_2: wartość;
    ...
}
```

15 Ciekawa i wprowadzająca pewną interaktywność jest możliwość określenia stylu dla znacznika tylko wtedy, gdy kursor myszy znajduje się na jego obszarze. Jest to często wykorzystywane w przypadku przycisków. By nadać znacznikowi wygląd specyficzny dla momentu, gdy znajduje się na nim kursor, po nazwie znacznika i dwukropku piszemy **hover**. Oto schemat:

```
nazwa_znacznika:hover
{
    właściwość_1: wartość;
    właściwość_2: wartość;
    ...
}
```

16 Na wielu stronach linki, które nie zostały jeszcze odwiedzone, wyglądają inaczej niż te, na które kliknięto. By nadać różny wygląd linkom przed kliknięciem na nie i po kliknięciu, po znaczniku **a** (znacznik **a** odpowiada za dodawanie linków) w selektorze dajemy dwukropek i piszemy **link**. Taki selektor oznacza link, na który jeszcze nie kliknięto. Gdy **link** zamienimy na **visited**, nadamy styl tylko linkom odwiedzonym.

Właściwości CSS

Wiemy już, w jaki sposób wyselekcjonować znaczniki. Teraz zastanówmy się, co można zrobić z wyselekcjonowanymi znacznikami.
CSS pozwala na nadawanie różnym właściwościom znaczników nowych wartości. Zajmijmy się tym, jakie mogą to być właściwości i jakie wartości można im nadać. Poznajmy najpopularniejsze z nich.

Właściwości dotyczące tekstu

1 W wielu popularnych edytorach tekstowych są przyciski pozwalające na wyrównanie tekstu. Tekst na stronie internetowej również możemy wyrównywać, i to



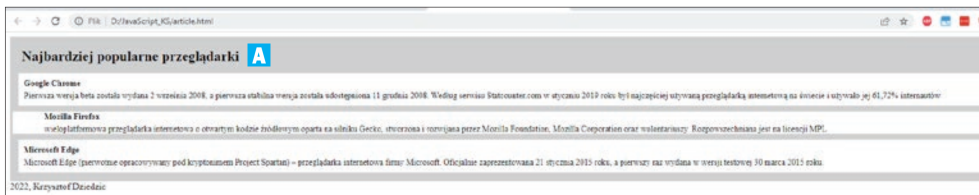
podobnie, jakbyśmy klikali na przyciski w edytorach tekstu. Służy do tego właściwość **text-align**.
Nadając jej wartość **center**, uzyskujemy teksty wyśrodkowane. Wartość **left** sprawi, że lewa kwawędź tekstu będzie równa, a prawa – skończy się, gdy zabraknie miejsca na kolejny wyraz. Wartość **right** przesunie tekst z wyrównaniem do prawej krawędzi. Właściwość ta ma jeszcze jedną możliwą wartość – **justyfy**, czyli wyjustowanie tekstu – prawa i lewa krawędź tekstu są równe, różna jest wielkość odstępów pomiędzy słowami.

2 Inną właściwością dotyczącą tekstu jest **text-indent**. Właściwość ta odpowiada

MARGINESY

SCHEMAT	OPIS
{ margin: wartość1; }	Jeśli do właściwości **margin** zostanie przypisana tylko jedna wartość, zostanie ustawiony margines z wszystkich czterech stron treści równy podanej wartości.
{ margin: wartość1 wartość2; }	Jeżeli dla właściwości **margin** zostaną podane dwie wartości, to pierwsza z nich odpowiada za wielkość marginesu górnego i dolnego, a druga – za wielkość marginesu prawego i lewego.
{ margin: wartość1 wartość2 wartość3; }	W przypadku podania trzech wartości pierwsza odnosi się do marginesu górnego, druga jest wspólna dla marginesu prawego i lewego, a ostatnia dotyczy marginesu dolnego.
{ margin: wartość1 wartość2 wartość3 wartość4; }	Podanie dla właściwości **margin** czterech wartości pozwoli na ustawienie różnych marginesów dla każdej z czterech krawędzi. Pierwsza wartość to margines górny, druga prawy, trzecia dolny, a czwarta lewy.
{ margin-top: wartość; }	By ustawić jedynie górny margines, możemy nadać wartość właściwości **margin-top**.
{ margin-left: wartość; }	Nadanie wartości dla właściwości **margin-left** pozwoli na ustawienie marginesu lewego.
{ margin-right: wartość; }	Nadanie wartości dla właściwości **margin-right** pozwoli na ustawienie marginesu prawego.
{ margin-bottom: wartość; }	Dolny margines ustawimy poprzez nadanie wartości dla właściwości **margin-bottom**.

adam13zajac@gmail.com



za wielkość wcięcia w pierwszym wierszu tekstu elementu, dla którego jest ustawiona. Wartość tej właściwości może być wyrażona między innymi w pikselach. I tak zapis **text-indent: 30px;** może dać następujący efekt **A**.

3 Ważną właściwością jest też **font-size**. To rozmiar czcionki. Wartość dla tej właściwości także możemy określać w pikselach.

4 Nie tylko wielkość czcionki, ale również jej rodzaj może być określany poprzez CSS. Służy do tego właściwość **font-family**.

Marginesy

Do ustawiania marginesów dla treści służy właściwość **margin**. Marginesy można ustawiać ze wszystkich czterech stron treści. Można nadawać wartość tej właściwości na różne sposoby.

Kolory

Ważnym elementem CSS jest możliwość kolorowania, na przykład tekstu czy tła.

1 Aby pokolorować tekst, nadaje się wartość właściwości **color**.

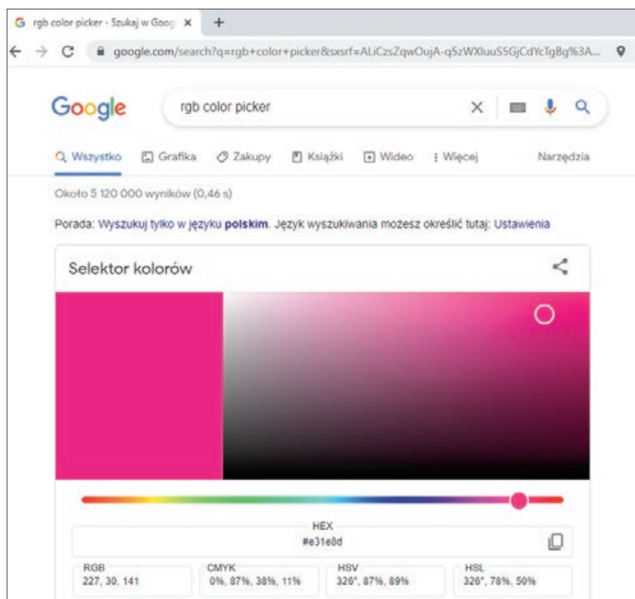
2 By ustawić kolor tła dla wybranej sekcji dokumentu, należy przypisać kolor do właściwości **background-color**.
Kolory najprościej jest określać angielskimi nazwami. Większość przeglądarek rozpoznaje nazwy 140 podstawowych kolorów. Ich listę można sprawdzić pod adresem **w3schools.com/cssref/css_colors.asp** **B**.

3 Co robić w sytuacji, gdy chcemy nadać kolor, który nie występuje na liście? Oczywiście i na to są różne sposoby. Jednym z nich jest model przestrzeni barw RGB. Pozwala on na tworzenie kolorów

CSS Colors
w3schools.com/cssref/css_colors.asp
HTML CSS JAVASCRIPT SQL PYTHON PHP BOOTSTRAP HOW TO W3.CSS JAVA
CSS Reference
CSS Reference
CSS Browser Support
CSS Selectors
CSS Functions
CSS Reference Aural
CSS Web Safe Fonts
CSS Fallback Fonts
CSS Animatable
CSS Units
CSS PX-EM Converter
CSS Colors
CSS Color Values
CSS Default Values
CSS Entities
CSS Properties
accent-color
align-content
align-items
align-self
all
animation
animation-delay
animation-direction
animation-duration
animation-fill-mode
animation-iteration-count
animation-name

DarkViolet #9400D3	DeepPink #FF1493	DeepSkyBlue #00BFFF
DimGray #696969	DimGrey #696969	DodgerBlue #1E90FF
FireBrick #B22222	FloralWhite #FFFAF0	ForestGreen #228B22
Fuchsia #FF00FF	Gainsboro #DCDCDC	GhostWhite #F8F8FF
Gold #FFD700	GoldenRod #DAA520	Gray #808080
Grey #808080	Green #008000	GreenYellow #ADFF2F

B



poprzez podanie trzech wartości liczbowych, odpowiadających barwom: czerwonej (**R**ed), zielonej (**G**reen) i niebieskiej (**B**lue). Kolory utworzone w ten sposób zapisujemy według schematu:
rgb(wartość_czerwonego, wartość_zielonego, wartość_niebieskiego).
Istnieje wiele narzędzi pozwalających na wybieranie kolorów z palety i sprawdzanie ich zapisu w RGB. Takie narzędzie udostępnia na przykład wyszukiwarka Google – by z niego skorzystać, należy w nią wpisać frazę **rgb color picker**.

Jednostki wielkości

Właściwościom odpowiedzialnym za wielkości można nadawać wartości w różnych jednostkach. Zwykle właściwości takie jak wysokość (**height**), szerokość (**width**) czy margines wyraża się w pikselach. To jednak nie jest obligatoryjne. Opis najważniejszych jednostek zawiera tabela.

JEDNOSTKI

SKRÓT	OPIS
px	**piksel** – ta miara odnosi się do urządzenia wyświetlającego. W przypadku urządzeń o niskiej rozdzielczości 1 piksel odpowiada jednemu pikselowi (kropce) wyświetlacza. W przypadku dużych rozdzielczości 1 piksel może oznaczać wiele pikseli wyświetlacza. Przyjmuje się, że wielkość jednego piksela to 1/96 cala.
cm	**centymetr** – co ciekawe, wielkości można wyrażać też w innych miarach, jak właśnie centymetry.
mm	**milimetr** – jeśli chcemy być dokładniejsi, zamiast centymetrów użyjmy milimetrów.
in	**cal** – wielkości można wyrażać nie tylko w centymetrach i milimetrach, ale także w calach. Jeden cal to 2,54 cm.
pt	**punkt** – 1 pt to 1/72 cala.
%	Ważnym sposobem wyrażania wielkości jest podawanie wartości procentowej. Wielkość stanowi wtedy określony procent wielkości rodzica.

JavaScript – podsumowanie

Omówiliśmy już najważniejsze zagadnienia dotyczące HTML-a i CSS – teraz pora na JavaScript. Podsumujmy wiadomości na jego temat. Przede wszystkim: do dodawania skryptów do dokumentu HTML służy znacznik **script**.

```
<script>

</script>
```

Zmienne, stałe i tablice

1 Zaczniemy od czegoś, bez czego trudno sobie wyobrazić programowanie – od **zmiennych**.
Zmienne są jak pudełka na dane. Pozwalają na to, by pod określoną nazwą można było przechowywać jakąś wartość.
I tak na przykład możemy mieć zmienną o nazwie **punkty** i pod tą nazwą będzie się kryła konkretna wartość, czyli liczba punktów zdobytych przez gracza. W JavaScripcie, by utworzyć zmienną, należy ją zadeklarować. Zmienne deklarujemy, pisząc słowo **var**, a następnie podając **nazwę zmiennej**. Możliwe, choć nieobligatoryjne jest od razu nadanie jej wartości początkowej – robimy to po znaku równości. Można też zadeklarować zmienną bez podawania jej wartości początkowej, nadając ją dopiero później. Oto schemat:

```
var nazwa_zmiennej = wartość_początkowa;
```

2 Nie tylko zmienne pozwalają na przechowywanie pewnej zawartości pod określoną nazwą. Podobnie jest ze **stałymi**. Różnicą jest jednak to, że jak wskazuje na to nazwa, stała ma wartość, która nie zmienia się podczas wykonywania programu. Różnica w tworzeniu jest natomiast taka, że zamiast słowa **var** w przypadku stałych podaje się **const**. Oto schemat:

```
const nazwa_stałej = wartość_stałej;
```

3 A jeśli już mówimy o przechowywaniu, to warto wspomnieć też o czymś, co nazywamy **tablicami**. W przybliżeniu można powiedzieć, że jest to specjalny rodzaj zmiennych, które pozwalają na przechowywanie pod jedną nazwą więcej niż jednej wartości. Dane przechowywane w tablicach są ponumerowane, dzięki czemu możemy dostać się do każdej z wartości z osobna. By zadeklarować tablicę, podobnie jak w przypadku zmiennych po słowie **var** (lub **const**, jeśli zawartość tablicy ma być stała) podaje się jej nazwę. W ten sposób tworzymy zmienną. Następnie trzeba ją przerobić na tablicę. Stanie się ona tablicą, gdy jako wartość przypiszemy do zmiennej nawias kwadratowy. Może być pusty.
Następnie, by nadawać wartości elementom tablicy, przypisuje się wartości do nazwy tablicy, po której umieszcza się nawias kwadratowy z zawartym w nim numerem, pod jakim chcemy w tablicy umieścić konkretną wartość.
Podobnie jest z odczytywaniem wartości z tablicy – po nazwie podajemy kwadratowy nawias z liczbą (indeksem elementu w tablicy), by się do niego odnieść.
Jeśli chcemy od razu nadać wartości kolejnym elementom tablicy, to podczas jej deklarowania wartości odpowiadające kolejnym elementom (numerowane od zera) umieszczamy w nawiasie kwadratowym. Oto schemat:

```
var nazwa_tablicy = [];
nazwa_tablicy[0] = wartość_pierwszego_elementu;
nazwa_tablicy[1] = wartość_drugiego_elementu;
nazwa_tablicy[2] = wartość_trzeciego_elementu;
//lub
var nazwa_tablicy = [wartość_1, wartość_2, wartość_3, ...];
```

4 By dodać element do tablicy na jej końcu, można użyć polecenia **push**. Wywołujemy je dla tablicy o podanej nazwie, po kropce. Po nazwie polecenia, w nawiasie, podajemy wartość, którą dodajemy do tablicy. Oto schemat A.

```
nazwa_tablicy.push(wartość_do_umieszczenia_w_tablicy);
```
A

5 Innym poleceniem dotyczącym tablic jest **length**. Pozwala ono na odczytanie, ile elementów ma tablica. Wartość ta jest zawsze o jeden większa niż indeks ostatniego elementu w tablicy. Oto schemat:

```
nazwa_tablicy.length
```

Pętle

Pętle to w programowaniu konstrukcje pozwalające na wielokrotne wykonywanie wybranych instrukcji. Mamy różne typy pętli.

1 Jednym z nich jest pętla **while** – pozwala ona na wykonywanie określonych instrukcji, dopóki jest spełniony jakiś warunek. Oto schemat:

```
while (warunek)
{
    //blok instrukcji do wykonania w pętli
}
```

2 Innym typem jest natomiast pętla **for**, którą stosuje się, gdy chcemy wykonać jakieś instrukcje określoną liczbę razy. Ważnym elementem tej pętli jest **zmienna sterująca**. Deklaruje się ją na potrzeby pętli i to tak naprawdę jej wartość jest wyznacznikiem tego, czy pętla jest wykonywana, czy nie. Dodatkowo z wartości tej zmiennej możemy korzystać wewnątrz pętli. Oto schemat **B**.

Instrukcja warunkowa

1 Ważnym elementem programowania jest też instrukcja warunkowa. Pozwala ona na wykonywanie instrukcji w zależności od innych czynników, czyli od określonego warunku.
Instrukcję warunkową tworzymy, pisząc **if**, a następnie w nawiasie zapisując warunek, który ma być spełniony. Jeśli w wypadku spełnienia warunku ma się wykonać tylko jedna instrukcja, możemy napisać ją bezpośrednio po nawiasie z warunkiem.
Oto schemat:

```
if (warunek) instrukcja_do_wykonania;
```

2 Jeśli w wypadku spełnienia warunku chcemy, by wykonało się więcej instrukcji, blok instrukcji do wykonania umieszczamy w nawiasie klamrowym. Oto schemat **C**.

3 W jednej instrukcji warunkowej może znaleźć się więcej niż jeden warunek.

C
```
if (warunek)
{
    //blok instrukcji do wykonania przy spełnionym warunku
}
```

Różne instrukcje mogą być wykonane, gdy inne warunki są spełnione. Następne warunki umieszcza się w nawiasach po **else if**. Kolejne warunki są sprawdzane, gdy nie są spełnione wcześniejsze warunki. Oto schemat **D**.

B
```
for (var nazwa_zmiennej_sterującej = wartość_początkowa; warunek_końca; zmiana_zminnej_sterującej)
{
    //blok instrukcji do wykonania w pętli
}
```

D
```
if (warunek)
{
    //blok instrukcji do wykonania przy spełnionym pierwszym warunku
}
else if (warunek2)
{
    //blok instrukcji do wykonania przy spełnionym drugim warunku
}
```

```
if (warunek)
{
    //blok instrukcji do wykonania przy spełnionym pierwszym warunku
}
else if (warunek2)
{
    //blok instrukcji do wykonania przy spełnionym drugim warunku
}
else
{
    //blok instrukcji do wykonania, gdy nie żaden z warunków nie był spełniony
}
```

E

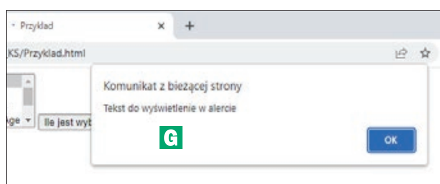
4 Instrukcja warunkowa może mieć też sekcję **else**. W niej opisujemy, co ma się wykonać, gdy żaden z warunków nie został spełniony. Oto schemat E.

Alert

To bardzo ciekawe polecenie. Stosujemy to polecenie, by przeglądarka wyświetliła wiadomość do użytkownika.
Treść wiadomości podaje się w nawiasie, po słowie alert. Przykładem wykonania wiersza **alert("Przykładowy alert!");** jest F. Oto komunikat G.

F

```
alert("Tekst do wyświetlenia w alercie");
```



ALTERNATYWNE IDE DO PROGRAMOWANIA

Oprócz Atom, którego używaliśmy we wskazówkach, warto wypróbować Visual Studio Code. To jeden z najpopularniejszych darmowych edytorów kodu źródłowego, stworzony przez Microsoft jako darmowe narzędzie z licencją MIT. Jest wysoce konfigurowalny, oferuje kolorowanie składni dla wielu języków, inteligentne uzupełnianie kodu, fragmentów, refaktoring kodu i wbudowany Git. Obsługuje różne motywy, skróty klawiaturowe i rozszerzenia, które dodają dodatkowe funkcje. Ma wersje na Windows, Linux i macOS.

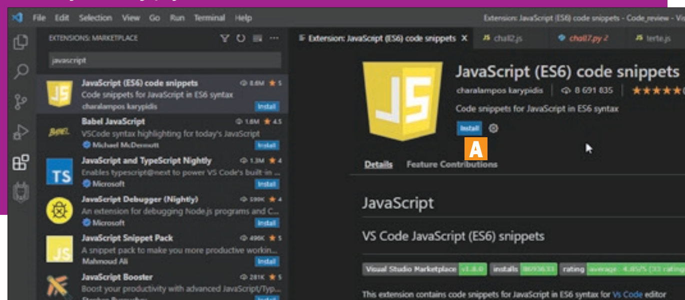
Programujemy w wybranym języku

Zanim zaczniemy pracę w Visual Studio Code, musimy sprawdzić, czy nasz język jest wspierany i czy można zainstalować dodatkowe narzędzia ułatwiające pracę.

1 Po uruchomieniu programu klikamy po prawej stronie na **Extensions**. Po lewej stronie na liście wyszukujemy język, który nas interesuje, i go wybieramy. Po prawej stronie znajdziemy szczegółowe informacje dotyczące konkretnego języka lub rozszerzenia, instalujemy go, klikając na **Install** A.

2 Tworzymy nowy dokument, klikając na **File**, **New File** i nadajemy mu nazwę.

3 Klikamy na **File**, **Save As** i nadajemy plikowi dowolną nazwę, a jako rozszerzenie podajemy **js** – dla plików JS. Dzięki temu dla wybranego pliku zostanie zastosowane odpowiednie podświetlanie składni.



JAK SKORZYSTAĆ Z E-WYDANIA KSIĄŻKI

W KŚ+ znajdziemy **e-wydanie tej Biblioteczki, obraz ISO dołączonej do niej płyty z najlepszymi narzędziami do programowania stron internetowych i plik PDF książki do pobrania.**

1 Otwieramy stronę **ksplus.pl**. Logujemy się **A** (używamy konta z serwisu **Komputerswiat.pl**). Jeżeli nie mamy konta, klikamy na **B**, by się zarejestrować.

B Załóż konto Logowanie A
Zarejestruj kod

2 Po zalogowaniu się możemy zarejestrować kod nadrukowany na płycie dołączonej do książki. Wystarczy kliknąć na **C** i przepisać kod.

Moje konto
C Zarejestruj kod

3 Uzyskamy w ten sposób dostęp do e-wydania **D** i do bonusowego obrazu płyty **E**. Do serwisu KŚ+ możemy logować się z dowolnego urządzenia z dostępem do internetu.

CZYTAJ E-WYDANIE D
PROGRAMY E
BONUSY

UWAGA! W KŚ+ ZA DARMO E-WYDANIE KSIĄŻKI ORAZ PLIK ISO PŁYTY



Krzysztof Dziedzic
autor książki, informatyk

JAK PROGRAMOWAĆ STRONY I APLIKACJE INTERNETOWE

Ta książka to wprowadzenie dla wszystkich, którzy planują rozpocząć przygodę z tworzeniem stron internetowych. Znajdują się w niej treści dla zupełnie początkujących i już nieco bardziej zaawansowanych czytelników. Najbardziej skoncentrowałem się w niej na nauce JavaScriptu. Dodatkowo przedstawiam też inne technologie internetowe, takie jak HTML, CSS i PHP.
Dla zupełnie początkujących pokazuję krok po kroku, jak szybko i bez wiedzy technicznej w najprostszy sposób założyć własnego bloga w specjalnym serwisie.
Nieco wyższy stopień wtajemniczenia to stworzenie bardziej rozbudowanej strony z wykorzystaniem funkcjonalności oferowanych przez WordPressa. W książce znajdziemy instrukcje, jak krok po kroku lokalnie postawić serwer, zainstalować bazę danych na serwerze PHP i zintegrować wszystko z naszą witryną. Dzięki WordPressowi i ogromnej ilości dodatków wszystkie te operacje wykonamy w chwilę i będziemy mogli rozpocząć tworzenie i testowanie naszej witryny. W przypadku tworzenia stron w WordPressie w podstawowych zastosowaniach nie musimy mieć wiedzy o zaawansowanych technologiach internetowych.
W dalszej części książki skupiłem się na JavaScripcie. Zaczynam od podstaw, wyjaśniając proces programowania w tym języku skryptowym. Następnie pokazuję, jak integrować skrypty ze stronami internetowymi. Na bazie podstawowych wiadomości, krok po kroku przedstawiam, jak napisać prostą grę przeglądarkową w JavaScripcie. Potem, również krok po kroku, opisuję, jak stworzyć własną aplikację webową, która będzie wyświetlać dane o aktualnej pogodzie. W tym celu wykorzystamy API z zewnętrznego serwera i wykonamy żądania AJAX.
Na płycie dołączonej do książki znajdziemy narzędzia przydatne do wykonania wskazówek, a z KŚ+ (ksplus.pl) można pobrać gotowe pliki z omówionymi projektami.

CENA 18,90 ZŁ
W TYM 5% VAT

Płyta DVD jest dodatkiem do książki

ISBN 978-83-8250-195-7 INDEKS 321 958



Nr 4/2022 (120)



KOMPUTER ŚWIAT BIBLIOTECZKA